Technická univerzita v Liberci

Fakulta mechatroniky a mezioborových inženýrských studií

Katedra softwarového inženýrství

Diplomová práce

Vybrané algoritmy optimálního řízení robotů

Selected algorithms of optimum robot control

Liberec 2004 Jakub Štilec

Anotace

Diplomová práce se zabývá dynamickými metodami optimálního řízení založenými na variačním počtu. Práce je rozdělena na pět částí. První z nich popisuje základní problém optimálního řízení. Dále obsahuje popis gradientní metody 1. řádu a metody nelineární střelby, které byly v rámci této práce implementovány v prostředí Visual C++. V další kapitole je ověření obou metod na jednoduché úloze a porovnání výsledků s analytickým řešením, získaným pomocí variačního počtu. Následující část obsahuje ověření obou metod při řešení základních struktur robotů (cylindrický, sférický, angulární a SCARA robot) a také způsoby nastavení vhodného počátečního odhadu. V předposlední části je popsána úprava gradientní metody 1. řádu pro řešení úlohy optimálního řízení v prostoru se statickou překážkou. Poslední část obsahuje realizaci simulačního prostředí pro zobrazení virtuálních modelů základních struktur robotů ve 3D, opět implementované v prostředí Visual C++ s využitím systému OpenGL. Veškeré programy vytvořené v rámci této práce v prostředí Visual C++ 7.0 včetně jejich zdrojových kódů jsou obsaženy na přiloženém CD-ROM.

Summary

This diploma thesis deals with dynamic methods of optimal control based on calculus of variations. The text is divided into five parts. The first part describes the crucial issue of optimal control. This part also contains a description of the first order gradient method and the nonlinear shooting method, which were implemented in Visual C++ 7.0 within the framework of this diploma thesis. The next chapter deals with verifying these methods on a simple task and with comparing the results to the analytic solution gained through the calculus of variations. The following part contains the verification of both methods on the fundamental structures of robots (cylindrical, spherical, angular and SCARA robot) and the ways of setting a suitable initial estimation. The penultimate part of this diploma thesis describes modification of the first order gradient method for solving the task of optimal control in space with static obstacles. The last chapter describes creation of simulation environment for visualization of virtual models of fundamental structures of robots in 3D. This simulation environment was also created in Visual C++ using the OpenGL system. All the programs created within the framework of this diploma thesis in Visual C++ 7.0 together with their source codes are included in the enclosed CD-ROM.

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé DP a prohlašuji, že souhlasím s případným využitím mé diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na náhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné velikosti).

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury na základě konzultací s vedoucím diplomové práce a konzultantem.

V Liberci dne 17. května 2004

…….

Jakub Štilec

Poděkování

Rád bych poděkoval vedoucímu diplomové práce Doc. Mgr. Ing. Václavu Zádovi, CSc. za vedení, cenné rady a celkovou podporu při tvorbě této diplomové práce.

Obsah

1. Úvod

Již v minulosti řada matematiků a přírodovědců dospěla k závěru, že veškeré děje v přírodě se chovají „optimálně“, neboli že příroda je optimálním systémem. Právě proto byly optimalizační úlohy předmětem matematického zkoumání již od počátků rozvoje matematiky a také byla snaha o vytvoření obecných postupů pro řešení těchto extremálních úloh.

Základ teorie optimálního řízení tvoří variační počet [1], který se jako samostatné odvětví matematiky začal rozvíjet od roku 1696. Tehdy Bernoulli poprvé upozornil na „princip optimality“, když formuloval úlohu o brachystochroně. Jednalo se o nalezení „nejlepší“ křivky spojující body A, B takové, že hmotný bod pohybující se po této křivce z bodu A pouze působením své vlastní tíže s nulovou počáteční rychlostí, dostane do bodu B za nejkratší možný čas. Variační počet pak byl rozpracován Eulerem, Lagrangem, Weierstrassem a dalšími. V průběhu roku 1950 byl v USA variační počet aplikován na základní problémy optimálního řízení. V roce 1958 Pontryagin objevil princip maxima, založený na variačním počtu. Optimální řízení se v těchto letech stalo samostatným vědním oborem, jehož uplatnění bylo především ve vesmírném výzkumu a v letectví.

Každý problém řízení má velké množství řešení a optimální řízení nám zavádí systematické metody, které vedou k „nejlepšímu“ řešení uvažovaného problému, podle námi zvoleného kritéria. Nejčastěji uvažujeme jako kritérium minimalizace čas potřebný k provedení pohybu, nebo kvadratické kriterium spotřebované energie. Takovéto řešení je velmi výhodné a to především proto, že při minimalizaci spotřebované energie snížíme celkové náklady a také zvýšíme životnost řízených systémů. To proto, že se v optimálních průbězích prudké a rázové pohyby, které životnost systémů snižují, vůbec nevyskytují, nebo se vyskytují pouze velmi málo.

Ve druhé části této práce je uveden základní problém optimálního řízení [2]. Dále je zde popsána gradientní metoda 1. řádu, jejíž popis byl získán od vedoucího této práce a metoda nelineární střelby, jejíž hlavní část byla převzata z [2]. Dále je zde uvedena vlastní implementace těchto metod ve vývojovém prostředí Visual C++ 7.0 a popis hlavních matematických metod, které bylo nutné implementovat. Na konci druhé části je uveden

popis metody optimálního řízení robotů, v minulosti realizované na katedře softwarového inženýrství a její porovnání s dynamickými optimalizačními metodami řešenými v této práci.

Ve třetí části naleznete analytické řešení jednoduché úlohy, kterou jsme pro tuto práci formulovali, pomocí metod variačního počtu [1]. Dále pak řešení této úlohy pomocí vlastních realizací gradientní metody 1. řádu a metody nelineární střelby. Jejich vzájemné porovnání a porovnání výsledků obou metod s analytickým řešením.

Ve čtvrté části je uveden popis řešení základních struktur robotů (cylindrický, sférický, angulární a SCARA robot) pomocí vlastních realizací gradientní metody 1. řádu a metody nelineární střelby. Dále je zde vzájemné porovnání výsledků získaných pomocí těchto metod spolu s popisem způsobu nastavení vhodného počátečního odhadu. Pohybové rovnice základních struktur robotů spolu s jejich parametry byly získány od vedoucího této práce.

Pátá část obsahuje úvahu o možnosti úpravy gradientní metody 1. řádu pro optimální řízení v prostoru se statickými překážkami. Dále je zde formulován jednoduchý systém se dvěma stupni volnosti a popsána vlastní realizace úpravy gradientní metody 1. řádu pro řízení v prostoru se statickými překážkami.

Poslední šestá část obsahuje způsob vizualizace základních struktur robotů. Dále pak popis základních vlastností systému OpenGL [8], vlastní implementaci vizualizace v prostředí Visual C++ a popis způsobu vytvoření jednotlivých virtuálních objektů a datových struktur pro jejich uložení.

Část příloh obsahuje grafické porovnání výsledků řešení jednotlivých úloh pomocí obou metod. V příloze číslo 6 je uveden manuál k aplikaci realizované v rámci této diplomové práce. Samotná aplikace, spolu se zdrojovými kódy je obsažena na přiloženém CD-ROM.

Veškeré výpočty uvedené v této diplomové práci byly provedeny na počítači s procesorem Intel Pentium 4 s frekvencí 2.53GHz, pamětí RAM 512MB a operačním systémem Windows XP.

2. Metody optimálního řízení

Problémem optimálního řízení, je najít takové řízení, které převede daný systém do nového stavu a to s podmínkou, že toto řešení bude ze všech možných přípustných řešení nejlepší, podle námi zvoleného kritéria. V této práci se budeme zabývat pouze dynamickými metodami optimalizace založenými na variačním počtu a typem kritéria, které minimalizuje velikosti jednotlivých řídících veličin, tedy minimalizuje energii nutnou k dosažení cílového stavu systému.

2.1. Definice problému optimálního řízení

V této kapitole budeme postupovat podle [2]. Aby jsme byli schopni najít vhodné řešení, musíme nejprve problém optimálního řízení popsat. Tedy sestavit dynamické rovnice, které chování systému nejlépe vystihují a z nich pak již jednoduše sestavit stavový popis systému ve tvaru:

$$\dot{x} = f\left(t, x, u\right). \tag{2.1}$$

Čas t je jako jediný parametr skalár, ostatní složky výrazu jsou vektory v následujícím tvaru:

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \qquad f = \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}, \qquad u = \begin{bmatrix} u_1 \\ \vdots \\ u_m \end{bmatrix}. \tag{2.2}$$

Dále předpokládejme, že funkce f je spojitá a že spojité jsou také její parciální derivace podle všech svých proměnných $x_1, \ldots, x_n, u_1, \ldots, u_m$ a času t.

Také musíme stanovit okrajové podmínky definující počáteční stav systému v čase t_0 a požadovaný stav systému v koncovém čase t_1 (tedy hodnoty stavového vektoru $x(t)$ v těchto časech). Zvolme funkci splnění okrajových podmínek v koncovém čase jako:

$$G\left(x\left(t_1\right)\right) = \begin{bmatrix} g_1\left(x\left(t_1\right)\right) \\ \cdots \\ g_n\left(x\left(t_1\right)\right) \end{bmatrix}, \tag{2.3}$$

kde jednotlivé funkce $g_i(t)$ vyjadřují velikost odchylky *i*-té stavové proměnné od okrajových podmínek v čase t_1. Z tohoto vztahu je dobře vidět, že *i*-tá odchylka není závislá pouze na hodnotě *i*-té stavové proměnné, ale na hodnotách celého vektoru *x*. Této vlastnosti využijeme v případě, kdy řešíme úlohu, kde koncový stav není definovaný číselně, ale pomocí nějaké analytické funkce. Např. pokud budeme uvažovat stavový systém o dvou stavových proměnných, pak polohu koncového stavu můžeme požadovat např. na kružnici.

Výpočet probíhá v jednotlivých iteracích (o celkovém počtu *N*) od počátečních podmínek $x(t_0)$ v čase t_0 do času t_1, kdy musí být splněna následující podmínka:

$$\lim_{N\to\infty} G\left(x\left(t_1\right)\right) = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{2.4}$$

Nyní je ještě nutné definovat podle čeho budeme optimalizovat. Ve funkci kritéria je možné mimo hodnot řízení také samozřejmě použít hodnoty jednotlivých stavových proměnných a optimalizovat pak podle nich. Funkce kritéria vyjádřená vztahem (2.5) má dvě části. První z nich $\Phi(t_1)$ je pevná, nezávislá na přechodu mezi počátečním a koncovým stavem. Tato část závisí pouze na koncovém stavu a penalizuje jeho odchylku od okrajových podmínek. Druhá integrální část již závisí na průběhu přechodu mezi počátečním a koncovým stavem. Hodnota funkce kritéria má tedy tvar:

$$J(t,x,u) = \phi(t_1) + \int_{t_0}^{t_1} f_0(t,x(t),u(t))dt\,. \tag{2.5}$$

Problémem optimálního řízení je převést systém ze stavu $x(t_0)$ v počátečním čase t_0 do stavu $x(t_1)$ v koncovém čase t_1, tak aby hodnota funkce kritéria (2.5) byla co nejmenší. Tedy hledáme takový vektor řízení $u(t)$ definovaný na intervalu $\langle t_0, t_1\rangle$, pro který je splněna následující podmínka:

$$J^*(t,x) = \min_u(J(t,x,u))\,, \tag{2.6}$$

kde J^* odpovídá minimální možné dosažitelné hodnotě kritéria. Funkce kritéria nabývá pro toto řízení nejmenší možné hodnoty a získané řešení pak nazýváme extremální.

Zavádíme skalární funkci - Hamiltonián:

$$H(\lambda, x, u) = f_0(x,u) + \lambda^T \cdot f(x,u) \tag{2.7}$$

a dále také Hamiltonovy kanonické rovnice:

$$\dot{x} = \left(\frac{dH}{d\lambda}\right)^T, \ \dot{\lambda} = -\left(\frac{dH}{dx}\right)^T. \tag{2.8}$$

Tyto rovnice mají schopnost předvídat účinky variací $\delta x(t)$ v čase t na stav systému v konečném čase v čase t_1.

K popisu principu numerických metod použitých k řešení úloh optimálního řízení je ještě nutné zmínit Pontrjaginův princip minima. Nechť $u(t)$ je přípustné řízení a $x(t)$ je mu odpovídající průběh stavových proměnných v čase t a nechť jsou definovány okrajové podmínky. K tomu, aby $u(t)$ a $x(t)$ byly optimálními průběhy na intervalu $\langle t_0, t_1 \rangle$, je nutné, aby existovalo takové netriviální řešení $\lambda(t)$ adjungované rovnice

$$\dot{\lambda} = -\left(\frac{dH}{dx}\right)^T, \tag{2.9}$$

že Hamiltonián $H(\lambda(t), x(t), u(t))$ dosahuje pro každé t z intervalu $\langle t_0, t_1 \rangle$, které je bodem spojitosti $u(t)$, svého minima stejně jako funkce u.

$$H^*(x,u,t) = \min_{\lambda}(H(\lambda, x, u, t)), \tag{2.10}$$

$$\left(\frac{dH}{du}\right)^T = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{2.11}$$

Pontrjaginův princip udává nutnou, ne však postačující podmínku optimality. Funkce $u(t)$ a $x(t)$, které splňují tuto podmínku se po řadě nazývají extremální řízení a extrémální trajektorie (extrémála). Optimální řízení, pokud existují, musí být mezi extremálními.

2.2. Popis gradientní metody 1. řádu

Gradientní metoda 1. řádu využívá k minimalizaci funkce kritéria J iterační postup. Vychází ze zvoleného nominálního řízení $u_0(t)$ definované pro t z intervalu $\langle t_0, t_1 \rangle$. Tento počáteční odhad řízení v jednotlivých iteracích upravuje a vzniká tak posloupnost řízení

$u_0, \ldots, u_n$ a k nim odpovídající posloupnost hodnot funkce kritéria $J_0, \ldots, J_n$. Tato posloupnost má záporný gradientní spád (klesající charakter), odtud je také odvozen název této metody.

Výhodou metody je integrace jednotlivých funkcí vždy ve stabilním směru, tedy dopředná integrace stavových rovnic (2.1) a zpětná integrace adjungovaných rovnic (2.9). Tento způsob řešení výrazně zvyšuje stabilitu výpočtu.

Nyní si popíšeme jednotlivé kroky tohoto algoritmu, který jsme převzali s drobnými úpravami z [2]:

1. Zvolíme nominální řízení $u_0(t)$ definované pro t v intervalu $\langle t_0, t_1 \rangle$. U složitějších a silně nelineární systémů bude na této volbě záviset úspěšnost a rychlost celého výpočtu.
2. Integrujeme stavové rovnice (2.1) s počáteční podmínkou $x(t_0) = x_0$. V průběhu integrace ukládáme průběhy $x(t)$ a také hodnoty parciálních derivací funkcí f, f_0 podle všech proměnných v čase t.
3. Vypočteme hodnotu penalizační funkce $\Phi[x(t_1)]$. Dále pak určíme hodnotu počátečních podmínek pro zpětnou integraci adjungovaných rovnic:

$$\lambda^T(t_1) = \frac{\partial \Phi}{\partial x}(t_1). \tag{2.12}$$

4. Integrujeme adjungované rovnice (2.9) zpětně v čase s počáteční podmínkou (2.11). Při zpětné integraci pro výpočet využíváme hodnoty uložené v kroku 2 algoritmu.
5. Provedeme výběr nejvhodnějšího koeficientu α. Metodu tohoto výběru popíšeme později.
6. Vypočítáme změnu řízení $\delta u(t)$ a v i-té iteraci vytvoříme řízení nové $u_i(t)$ podle vztahů:

$$\delta u(t) = -\alpha \frac{\partial H^T}{\partial u}(t), \; u_i(t) = u_{i-1}(t) + \delta u(t). \tag{2.13}$$

7. Pokud jsou řídící veličiny nějakým způsobem omezeny, pak v tomto kroku jednoduše aplikujeme omezení:

$$u_i(t) = \begin{cases} u_{\min} & pokud \quad u_i(t) < u_{\min} \\ u_{\max} & pokud \quad u_i(t) > u_{\max} \\ u_i(t) & jinak \end{cases}. \tag{2.14}$$

8. Pokud je změna hodnoty kritéria ΔJ dostatečně malá, pak algoritmus ukončíme. V opačném případě pokračujeme krokem 2 algoritmu.

Tato metoda se projevuje největší změnou hodnot řízení *u(t)* a kritéria *J* v několika prvních iteracích a to i přes velmi špatný počáteční odhad. Její velkou nevýhodou však je, že konvergence k řešení je velmi pomalá.

2.3. Popis metody nelineární střelby

Metoda nelineární střelby využívá k minimalizaci funkce kritéria *J* stejně jako předcházející metoda iterační postup. Na rozdíl od gradientní metody 1. řádu nevyužívá počáteční odhad řízení $u_0(t)$ v čase *t* z intervalu $\langle t_0, t_1 \rangle$, ale počáteční odhad vektoru $\lambda(t_0)$, který má počet složek stejný, jako je počet stavových proměnných. Tento vektor je pak v jednotlivých iteracích postupně upravován, až je v *n*-té iteraci získáno řešení, které splňuje okrajové podmínky řešení dané úlohy.

Nevýhodou této metody je integrace jednotlivých funkcí vždy v přímém směru, tedy dopředná integrace stavových (2.1) i adjungovaných (2.9) rovnic a právě dopředná integrace adjungovaných rovnic není vždy stabilní a výpočet při volbě špatných okrajových podmínek selhává.

Nyní si popíšeme jednotlivé kroky tohoto algoritmu jehož základní kostru jsme získali z [2], ale provedli jsme velké množství modifikací:

1. Zvolíme počáteční odhad vektoru $\lambda(t_0) = \lambda_0$ v čase t_0 a celkový počet iterací *N*. Tato metoda je na počáteční odhad výrazně citlivější než gradientní metoda 1. řádu a špatný počáteční odhad může vést k selhání celého výpočtu.
2. Zavedeme funkci *theta* s parametrem $\lambda(t_0)$. Tato funkce najednou integruje stavové rovnice (2.1) s počáteční podmínkou $x(t_0) = x_0$ a adjungované rovnice (2.9) s počáteční podmínkou $\lambda(t_0)$ od počátečního času t_0 do koncového času t_1.

V průběhu integrace určujeme hodnoty řízení *u(t)* ze vztahu (2.11). Výstupem této funkce je vektor odchylek od splnění okrajové podmínky $G(x(t_1))$ ze vztahu (2.15).

3. Spustíme funkci theta pro počáteční odhad λ_0, vypočteme hodnotu $G_0 = G(x(t_1))$ a tuto hodnotu uložíme.
4. V *i*-tém iteračním kroku vypočteme hodnotu G_{iP} splnění okrajových podmínek které chceme v této iteraci dosáhnout (požadovaná hodnota, nikoliv vypočtená). Je vidět, že v nulté iteraci (počáteční odhad) platí $G_{iP} = G_0$ a v koncové iteraci je $G_{NP} = 0$. V jednotlivých iteracích mezi počáteční a konečnou je odchylka splnění okrajových podmínek interpolována. Samozřejmě můžeme použít i jiný typ funkce než je uvedeno ve vztahu (2.15), ale tento typ je jednoduchý na výpočet a pro většinu úloh plně dostačující.

$$G_{iP} = G_0 \left(1 - i/N\right). \tag{2.15}$$

5. Vypočteme odchylku splnění okrajových podmínek G_{i-1} zavoláním funkce theta s parametrem λ_{i-1}. Dále vypočteme vektor G_{DIF} podle vztahu:

$$G_{DIF} = G_{iP} - G_{i-1}. \tag{2.16}$$

6. Vypočteme Jakobián funkce theta jako čtvercovou matici s označením *A*, tedy matici parciálních derivací funkce G_{i-1} podle všech proměnných vektoru λ_i.
7. Nyní již můžeme vypočítat vektor úpravy $\delta\lambda$ a vypočítat nové λ_i podle vztahu:

$$\delta\lambda = A^{-1} G_{DIF}, \tag{2.17}$$

$$\lambda_i = \lambda_{i-1} + \delta\lambda. \tag{2.18}$$

8. Dále musíme zkontrolovat zda jsme pomocí nového λ_i řešení nepřestřelili. Proto pro něj spustíme funkci theta a pokud bude zjištěná odchylka od splnění okrajových podmínek G_i větší než hodnota již získaná G_{i-1}, pak zvolíme $\delta\lambda$ jako $\delta\lambda = \delta\lambda/2$ *a* λ_i opět vypočítáme podle vztahu (2.18). Takto pokračujeme tak dlouho, dokud neplatí $G_i < G_{i-1}$.
9. Pokud je G_i přibližně shodné s G_{iP} a krok iterace *i* je menší než celkový počet iterací *N* pak *i = i + 1* a pokračujeme další iterací krokem 4 algoritmu. Pokud je G_i přibližně shodné s G_{iP} a krok iterace *i = N* pak algoritmus končí. V případě, že se liší více než je definovaná tolerance, pak je nutné provést další doplňkový krok v rámci dané iterace. Položíme $\lambda_{i-1} = \lambda_i$ a pokračujeme krokem 5 algoritmu.

Pokud porovnáme jednotlivé kroky této metody s gradientní metodou 1. řádu, pak zjistíme že algoritmus této metody je výrazně složitější a výpočetně náročnější než gradientní metoda 1. řádu. Ovšem na druhou stranu je s dobrým počátečním odhadem schopen dojít k řešení výrazně rychleji a to s přibližně konstantním krokem konvergence. Velkou výhodou této metody je právě její rychlá konvergence.

2.4. Implementace jednotlivých metod

Obě výše zmíněné metody jsou metody iterační a k dosažení řešení je nutný velký počet výpočetních cyklů. Využíváme velké množství matematických metod jako je numerická integrace, parciální derivace, maticové výpočtu. Tyto operace jsou ve velkém množství využívány v každé iteraci a jsou výpočetně velmi náročné. Proto jsme se rozhodli realizovat obě metody v prostředí Visual C++ 7.0, které zaručí na rozdíl od ostatních vývojových prostředí (jako např. Delphi, nebo Java) vysokou rychlost a malou velikost vytvořené aplikace.

2.4.1. Implementace gradientní metody 1. řádu

Metodu jsme implementovali podle algoritmu zmíněného v části 2.2. Pro numerickou integraci jsme zvolili metodu Runge-Kutta 4. řádu s absolutní délkou kroku s úpravou pro možnost dopředného i zpětného směru integrace. Veškeré parciální derivace vyskytující se ve výpočtu jsou řešeny numericky.

Vstupem algoritmu jsou:

1. Stavový popis řešeného systému definovaný v (2.1).
2. Okrajové podmínky, tedy počáteční čas t_0 a koncový čas t_1 a hodnota stavového vektoru $x(t)$ v těchto časech.
3. Počet bodů rozlišení simulace b. Z konstanty b můžeme vypočítat velikost kroku simulace Δt podle následujícího vztahu:

$$\Delta t = \left(t_1 - t_0\right) / \left(b - 1\right). \tag{2.19}$$

4. Velikost konstanty α, která ovlivňuje velikost změny řízení v jednotlivých iteracích a její vhodné nastavení velmi ovlivní průběh celého výpočtu. Pokud je tato veličina příliš malá, výpočet je pomalý a pokud je naopak velká výpočet selhává.

V algoritmu je implementován automatický odhad, ale přesto je dobré určit vhodné počáteční nastavení ručně.

5. Velikost konstanty *K*. Tato konstanta penalizuje odchylku od okrajových hodnot. Pokud tuto hodnotu nastavíme příliš malou, algoritmus najde řešení, které nesplňuje okrajové podmínky. Pokud naopak nastavíme hodnotu příliš velkou algoritmus selhává. Obecně bylo zjištěno, že se zvyšující se velikostí konstanty *K* musíme snižovat velikost konstanty α.
6. Počáteční odhad řízení $u_0(t)$ definované pro *t* v intervalu $\langle t_0, t_1 \rangle$. Tento odhad je velmi důležitý, protože ovlivní běh celého výpočtu. U jednoduchých systémů s malým množstvím nelinearit vystačíme s nulovou konstantní funkcí. Zde se vhodným počátečním odhadem zabývat nemusíme. V případě složitějších systémů je již dobrý počáteční odhad nutný. Je třeba upozornit, že v případě špatného počátečního odhadu má algoritmus tendenci vyhledat nejbližší lokální minimum, které nám nemusí vyhovovat. Proto je třeba vždy s nastavením počátečního odhadu experimentovat.
7. Maximální počet provedených iterací, po kterých se běh programu zastaví.

Automatický odhad konstanty α byl proveden následujícím způsobem. Po stanoveném počtu iterací je tato konstanta vynásobena určitým číslem mírně vyšším než jedna. V každém kroku je řízení ukládáno do záložních proměnných a pokud je změna konstanty α nevyhovující (tento stav algoritmus rozpozná podle toho, že je gradient funkce kritéria *J* kladný), pak je obnoveno původní řízení a hodnota konstanty α vynásobena číslem nižším než jedna. Ukázalo se, že v průběhu výpočtu je možné tuto konstantu místy nepatrně zvýšit a takto urychlit výpočet.

Důležité je také zmínit typ funkcí pro penalizaci splnění okrajových podmínek. V tomto případě odpovídá hodnota penalizace $\Phi[x(t_1)]$ následujícímu vztahu:

$$\Phi\left(x\left(t_1\right)\right) = K \cdot G\left(x\left(t_1\right)\right). \tag{2.20}$$

Kde $G(x(t_1))$ je vektor ze vztahu (2.3), jehož jednotlivé složky $g_i(x(t_1))$ definujeme jako:

$$g_i\left(x\left(t_1\right)\right) = \left(x_i\left(t_1\right) - x_{iP}\left(t_1\right)\right)^2. \tag{2.21}$$

V tomto výrazu proměnná x_{ip} odpovídá požadované *i*-té hodnotě stavového vektoru v čase t_1. Hodnota počátečních podmínek pro zpětnou integraci adjungovaných rovnic má tedy následující tvar (zde $x(t_1)$ vystupuje jako stavový vektor a $x_P(t_1)$ je požadovaná hodnota stavového vektoru v koncovém čase):

$$\lambda^T\left(t_1\right) = 2K\left(x\left(t_1\right) - x_P\left(t_1\right)\right). \tag{2.22}$$

Výstupem algoritmu jsou následující parametry:

1. Průběhy stavového vektoru $x(t)$ pro t v intervalu $\langle t_0, t_1\rangle$.
2. Průběhy zrychlení $a(t)$ jednotlivých složek řízeného systému .
3. Průběhy jednotlivých řízení $u(t)$ systému.
4. Kritérium J získaného řešení.
5. Odchylka získaného řešení od okrajových podmínek, pro vyjádření této odchylky zavedeme následující tvar a budeme ji označovat pomocí proměnné *norma*:

$$norma = \sqrt{\sum_{i=1}^{n}\left(x_i\left(t_1\right) - x_i^*\left(t_1\right)\right)^2}\,. \tag{2.23}$$

Můžete si všimnout, že jsme nezapsali výsledné průběhy algoritmu jako optimální a to z toho důvodu, že uživatel může výpočet kdykoliv přerušit a výsledné průběhy, jsou pak pouze průběhy získané po *n* provedených iterací. Optimální řízení získáme až když změna kritéria ΔJ bude dostatečně malá a i přes provedení libovolného počtu dalších iterací bude řešení konvergovat ke stále stejné hodnotě. Pak nalézáme lokální extrém.

2.4.2. Implementace metody nelineární střelby

Metoda byla implementována podle algoritmu zmíněného v části 2.3. Pro numerickou integraci byla zvolena metoda Runge-Kutta 4. řádu s absolutní délkou kroku a na rozdíl od gradientní metody 1. řádu v této metodě stačila pouze dopředná integrace. Veškeré parciální derivace vyskytující se ve výpočtu jsou řešeny numericky. Tato metoda rovněž vyžaduje implementaci kódu pro výpočet inverzní matice.

Vstupem algoritmu jsou:

1. Okrajové podmínky, tedy počáteční čas t_0 a koncový čas t_1 a hodnota stavového vektoru $x(t)$ v těchto časech.

2. Počet bodů rozlišení simulace *b*. Z konstanty *b* můžeme vypočítat velikost kroku simulace Δt podle vztahu (2.19).
3. Celkový počet iterací *N*. U této metody konstanta *N* neurčuje pouze maximální počet iterací, ale ovlivňuje celkový charakter výpočtu, jak si můžete všimnout z popisu algoritmu v části 2.3.
4. Počáteční odhad vektoru $\lambda(t_0) = \lambda_0$ v čase t_0. Tento odhad má velký vliv na celý výpočet. U silně nelineárních systému je velice obtížné najít takový odhad, aby byl algoritmus schopen pracovat a neselhal.

Vzhledem k tomu, že tato metoda pracuje jiným způsobem než gradientní metoda, nemusíme penalizovat splnění okrajových podmínek a sám charakter této metody nám umožňuje dospět ke správnému řešení. Okrajové podmínky jsou opět charakterizovány pomocí funkce $G(x(t_1))$ popsané ve vztahu (2.3). V tomto případě však jako jednotlivé složky $g_i(x(t_1))$ tohoto vektoru není vhodné použít kvadratickou funkci (i když je to možné), protože její použití by výpočet příliš zkomplikovalo, proto byla použita lineární funkce ve tvaru:

$$g_i\left(x\left(t_1\right)\right) = x_i\left(t_1\right) - x_{iP}\left(t_1\right). \tag{2.24}$$

V tomto výrazu proměnná x_{ip} odpovídá požadované *i*-té hodnotě stavového vektoru v čase t_1.

Výstupem algoritmu jsou následující parametry:

1. Průběhy stavového vektoru $x(t)$ pro t v intervalu $\langle t_0, t_1 \rangle$.
2. Průběhy zrychlení $a(t)$ jednotlivých složek řízeného systému .
3. Průběhy jednotlivých řízení $u(t)$ systému.
4. Hodnota vektoru λ, pro který je řešení získáno.
5. Kritérium *J* získaného řešení.
6. Odchylka získaného řešení od okrajových podmínek pomocí proměnné *norma* definované v (2.23).

2.5. Popis řešení hlavních matematických operací

2.5.1. Numerická integrace

Jak již bylo výše zmíněno, pro numerickou integraci jsme použili metodu Runge-Kutta čtvrtého řádu s absolutní délkou kroku. Tato metoda je určena pro řešení diferenciálních rovnic 1. řádu a pro svůj výpočet potřebuje znát pouze jeden předchozí stav, tedy y_{i+1} závisí pouze na y_i. Výpočetní náročnost této metody roste lineárně s počtem bodů rozlišení simulace.

Snažili jsme se o obecnou implementaci, proto má jako jeden ze svých vstupních parametrů ukazatel na funkci obsahující soustavu diferenciálních rovnic 1. řádu. Navíc je do funkce předáván obecný ukazatel typu *void*, který nám umožňuje přenést libovolnou datovou strukturu, nutnou pro potřeby výpočtu. Obecný tvar této metody je obsažen ve vztahu (2.25) a (2.26). Zde proměnná *s* udává řád této metody a vztah (2.25) je počítán pro *k=1,…,s*. Z něho je pak vypočítán další krok metody pomocí (2.26).

$$T_k = t_j + \alpha_k \Delta t\,, \qquad X_k = X_j + \Delta t \sum_{m=0}^{s} \beta_{k,m} F_m\,, \qquad F_k = f\left(T_k, X_k\right), \tag{2.25}$$

$$x_{j+1} = x_j + \Delta t \sum_{k=0}^{s} \gamma_k F_k\,. \tag{2.26}$$

Úprava této metody na zpětnou integraci v čase je následující:

$$T_k = t_{b-j} - \alpha_k \Delta t\,, \qquad X_k = X_{b-j} - \Delta t \sum_{m=0}^{s} \beta_{k,m} F_m\,, \qquad F_k = f\left(T_k, X_k\right), \tag{2.27}$$

$$x_{b-j-1} = x_{b-j} - \Delta t \sum_{k=0}^{s} \gamma_k F_k\,. \tag{2.28}$$

Hodnota Δt odpovídá velikosti kroku simulace ze vztahu (2.13) a proměnná *b* odpovídá počtu kroků simulace. Pro Runge-Kuttovu metodu čtvrtého řádu mají jednotlivé koeficienty následují hodnoty:

$$s = 3,\ \alpha_1 = 0.5,\ \alpha_2 = 0.5,\ \alpha_3 = 1,\ \beta_{1,0} = 0.5,\ \beta_{2,1} = 0.5,\ \beta_{3,2} = 1,\ \gamma_1 = \gamma_2 = 1/3,\ \gamma_3 = 1/6. \tag{2.29}$$

2.5.2. Výpočet inverzní matice

Pro maticové operace byla v jazyce C++ vytvořena třída, která podporuje základní maticové operace, jako je sčítaní a odčítání matic, vzájemné násobení matic, násobení matice reálným číslem, transpozice matice, výpočet adjungované a inverzní matice a také výpočet determinantu.

Inverzní matice byla řešena podle základních vztahů:

$$A^{-1} = \frac{1}{\det A} adj\left(A\right),\ adj\left(A\right) = \hat{A}^{T}, \tag{2.30}$$

$$\hat{A}_{ij} = \left(-1\right)^{i+j} \bar{A}_{ij}, \tag{2.31}$$

kde matice $\hat{A}$ se nazývá matice kofaktorů. Prvek $\bar{A}_{ij}$ je minor, jedná se o determinant matice *A',* která vznikne z matice *A* vynecháním *i*-tého řádku a *j*-tého sloupce. Jednotlivé determinanty jsou vypočteny ze schodovitého tvaru dané matice, který je získán pomocí elementárních řádkových úprav matice s ohledem na změny v hodnotě determinantu, které vzniknou právě v jejich důsledku.

2.6. Alternativní metody optimálního řízení

V této části zmíníme metodu realizovanou na katedře softwarového inženýrství. Jedná se o aproximaci trajektorie pomocí spline funkce.

Numerické metody optimálního řízení jsou výpočetně velmi náročné. Proto byla snaha aproximovat pohyb systému pomocí vhodné známé funkce s konečným počtem parametrů, jejichž optimální hodnoty hledáme a která splňuje výraz (2.1). Tímto se celý problém převede na hledání minima reálné funkce s malým počtem proměnných. Takto sice získáme pouze sub-optimální řešení, je však zřejmé, že vhodnou volbou aproximující funkce a použitím dostatečného počtu proměnných se lze k optimálnímu řízení přiblížit. Jako optimalizační metody se pak používají parametrické optimalizační metody, jako jsou gradientní metody, metoda partan, flexibilní simplex, globální optimalizační metoda. Tímto způsobem se velice sníží počet výpočtů nutných k dosažení řešení na rozdíl od velké výpočetní náročnosti gradientní metody 1. řádu a metody nelineární střelby.

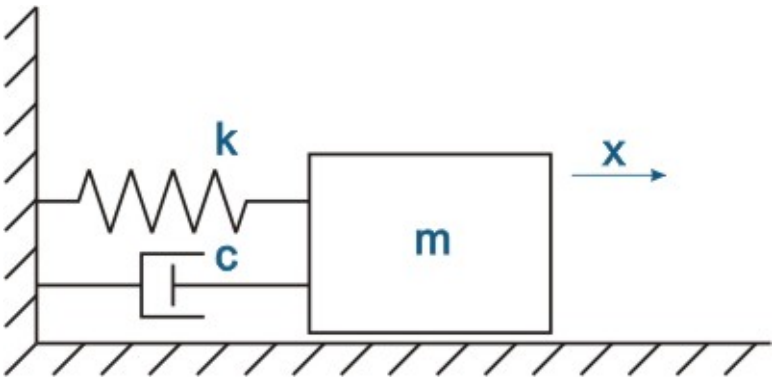
Nechť máme systém popsaný pomocí stavových rovnic (2.1) a jednotlivé funkce popisující tento systém jsou spojité i se svými prvními derivacemi. Jednotlivé stavové proměnné se v čase vyvíjejí a vytváří tak trajektorie. Principem této metody je rozdělení těchto trajektorií na *N* v čase ekvidistantních úseků. Vznikne tak posloupnost hraničních (uzlových) bodů o celkovém počtu *N+1*. Jednotlivé úseky se aproximují pomocí funkcí, které dostatečně přesně vystihují daný průběh. Pro aproximaci se využívá spline funkce. Tato funkce nám umožňuje spojitost (hladkost) v uzlových bodech až do *k*-tého řádu a splnění okrajových podmínek. Je nutné volit dostatečný počet parametrů, tak aby veškeré parametry nebyly vázané (zaručující spojitost v uzlových bodech a splnění okrajových podmínek), aby zde byly volitelné parametry pomocí kterých je možné průběh funkce optimalizovat.

Tento způsob řešení tedy spočívá ve výpočtu vhodných parametrů funkcí spline a následnou optimalizaci malého konečného počtu parametrů. Toto je velkou výhodou, protože umožňuje vysokou rychlost výpočtu a použití i globálních optimalizačních metod. Nevýhodou ovšem je nalezení pouze sub-optimálního řešení.

3. Řešení jednoduché úlohy

3.1. Formulace problému jednoduché úlohy

Aby jsme ověřili správnost realizovaných numerických metod, zvolili jsme nejprve jednoduchou úlohu, kterou jsme schopni řešit analyticky. Jako tuto úlohu jsme zvolili hmotný bod s jedním stupněm volnosti (posuvný pohyb ve směru osy *x*), na který působí pouze řídící síla *u* (působící ve směru osy x), síla pružiny a lineární tlumič rychlosti. Počátek osy *x* byl zvolen v rovnovážné poloze, kdy na hmotný bod působí nulová síla pružiny. Parametry úlohy: koeficient tuhosti $k_1 = 10Nm^{-1}$, koeficient tlumení $c_1 = 2Nm^{-1}s$, hmotnost $m = 1kg$. Úkolem je najít optimální řízení pro přesun hmotného bodu z polohy $x = 0m$ do polohy $x = 1m$ za čas $1s$, přičemž počáteční i koncová rychlost je nulová.



Obr. 3.1: Schéma jednoduché úlohy.

Řešení této úlohy začneme sestavením dynamického popisu systému:

$$m\ddot{x} = -c_1\dot{x} - k_1 x + u_1 \tag{3.1}$$

a k němu odpovídajícímu stavovému popisu systému. Systém popíšeme pomocí dvou stavových proměnných: poloha hmotného bodu x_1 a rychlost hmotného bodu x_2. Pro zjednodušení rovnic použijeme substituci $c = c_1 / m$, $k = k_1 / m$ a $u = u_1 / m$. Soustava stavových rovnic pak bude vypadat následovně:

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -cx_2 - kx_1 + u. \end{aligned} \tag{3.2}$$

Zbývá sestavit funkci kritéria, podle kterého budeme optimalizovat. V této úloze použijeme kvadratické kritérium řízení:

$$J = \int_{t_0}^{t_1} u^2 dt \,. \tag{3.3}$$

Také musíme formulovat okrajové podmínky řešení tohoto systému: počáteční čas $t_0 = 0s$, koncový čas $t_1 = 1s$, stav systému v počátečním čase $x_1(t_0) = 0m$, $x_2(t_0) = 0ms^{-1}$ a stav systému v koncovém čase $x_1(t_1) = 1m$, $x_2(t_1) = 0ms^{-1}$.

3.2. Analytické řešení jednoduché úlohy

Nyní máme náš problém definován a jako první ho budeme řešit analyticky pomocí metod variačního počtu [1]. Z rovnice (3.1) vyjádříme řízení u a dosadíme do rovnice kritéria (3.3), kde dostáváme:

$$J = \int_{t_0}^{t_1} (\ddot{x}^2 + 2c\ddot{x}\dot{x} + 2k\ddot{x}x + c^2\dot{x}^2 + 2ck\dot{x}x + k^2x^2)dt \,. \tag{3.4}$$

Extremální řešení této rovnice získáme řešením funkcionálu typu:

$$\upsilon\left[x(t)\right] = \int_{t_0}^{t_1} F\left(t, x(t), \dot{x}(t), \ddot{x}(t)\right)dt \text{, kde} \tag{3.5}$$

$$F\left(t, x(t), \dot{x}(t), \ddot{x}(t)\right) = \ddot{x}^2 + 2c\ddot{x}\dot{x} + 2k\ddot{x}x + c^2\dot{x}^2 + 2ck\dot{x}x + k^2x^2 \,. \tag{3.6}$$

Funkci $x(t)$ realizující extrém funkcionálu (3.5) obdržíme řešením Euler-Poissonovy rovnice čtvrtého řádu, jejíž obecný tvar je:

$$F_x - \frac{d}{dt}F_{\dot{x}} + \frac{d}{dt^2}F_{\ddot{x}} = 0 \,. \tag{3.7}$$

Vyjádříme-li,

$$\begin{aligned} F_x &= \frac{\partial}{\partial x}F\left(t, x(t), \dot{x}(t), \ddot{x}(t)\right) = 2k\ddot{x} + 2ck\dot{x} + 2k^2x \,, \\ F_{\dot{x}} &= \frac{\partial}{\partial \dot{x}}F\left(t, x(t), \dot{x}(t), \ddot{x}(t)\right) = 2c\ddot{x} + 2c^2\dot{x} + 2ckx \,, \\ F_{\ddot{x}} &= \frac{\partial}{\partial \ddot{x}}F\left(t, x(t), \dot{x}(t), \ddot{x}(t)\right) = 2\ddot{x} + 2c\dot{x} + 2kx \,, \end{aligned} \tag{3.8}$$

a dosadíme do rovnice (3.7), dostáváme Euler-Poissonovu rovnici :

$$x^{(4)} + 2cx^{(3)} + \left(c^2 + 2k\right)\ddot{x} + 2ck\dot{x} + k^2 x = 0\,. \tag{3.9}$$

Jedná se o homogenní lineární diferenciální rovnici čtvrtého řádu s konstantními koeficienty. Pro zadané parametry systému je její obecné řešení spolu s první a druhou derivací ve tvaru:

$$x(t) = C_1 e^{-t}\cos(3t) + C_2 e^{-t}\sin(3t) + C_3 e^{t}\cos(3t) + C_4 e^{t}\sin(3t)\,, \tag{3.10}$$

$$\begin{aligned}\dot{x}(t) = {} & C_1 e^{-t}\left(-\cos(3t) - 3\sin(3t)\right) + C_2 e^{-t}\left(-\sin(3t) + 3\cos(3t)\right) + \\ & + C_3 e^{t}\left(\cos(3t) - 3\sin(3t)\right) + C_4 e^{t}\left(\sin(3t) + 3\cos(3t)\right),\end{aligned} \tag{3.11}$$

$$\begin{aligned}\ddot{x}(t) = {} & C_1 e^{-t}\left(-8\cos(3t) + 6\sin(3t)\right) + C_2 e^{-t}\left(-6\cos(3t) - 8\sin(3t)\right) + \\ & + C_3 e^{t}\left(-8\cos(3t) - 6\sin(3t)\right) + C_4 e^{t}\left(6\cos(3t) - 8\sin(3t)\right).\end{aligned} \tag{3.12}$$

Konstanty C_1, C_2, C_3, C_4 určíme z okrajových podmínek, řešením soustavy čtyř rovnic o čtyřech neznámých:

$$x(0) = 0\,,\ \dot{x}(0) = 0\,,\ x(1) = 1,\ \dot{x}(1) = 0\,. \tag{3.13}$$

Pro dané parametry systému získáme:

$$\begin{aligned}& C_1 \doteq 0.395556\,,\ C_2 \doteq 0.065033\,, \\ & C_3 \doteq -0.395556\,,\ C_4 \doteq\ 0.198671.\end{aligned} \tag{3.14}$$

Z rovnice (3.1) vyjádříme hodnotu řízení *u* v čase *t* pro zadané parametry systému:

$$u(t) = \ddot{x}(t) + 2\dot{x}(t) + 10x(t)\,. \tag{3.15}$$

Po dosazení z rovnic (3.10-12) do rovnice (3.15) a následné úpravě dostáváme vztah:

$$u(t) = C_3 e^{t}\left(4\cos(3t) - 12\sin(3t)\right) + C_4 e^{t}\left(4\sin(3t) - 12\cos(3t)\right). \tag{3.16}$$

Hodnotu kritéria *J* získáme dosazením takto vyjádřené funkce řízení do rovnice (3.3). Po integraci a dosazení dostáváme (postup výpočtu vzhledem k jeho délce není uveden):

$$J = \int_0^1 \left[C_3 e^{t}\left(4\cos(3t) - 12\sin(3t)\right) + C_4 e^{t}\left(4\sin(3t) - 12\cos(3t)\right)\right]^2 dt\,, \tag{3.17}$$

$$J \doteq 45.627385\,. \tag{3.18}$$

Nyní je již tento jednoduchý systém kompletně analyticky vypočtený a můžeme na něm ověřit numerické metody.

3.3. Řešení pomocí gradientní metody 1. řádu

Počet bodů simulace byl nastaven na hodnotu 101, což odpovídá simulačnímu kroku o velikosti 0.01s. Odhad počátečního řízení jsme zvolili jako konstantní funkci *u(t) = 0*, konstantu *α = 1e-6* a penalizaci odchylky od konečného stavu *K = 1e7*. Při řešení této úlohy jsme nepoužili metodu automatického odhadu konstanty *α*, aby jsme mohli demonstrovat funkci samotné metody.

Po spuštění výpočtu, algoritmus v jednotlivých iteracích upravoval vektor řízení *u* a po 80 iteracích a 0.85s výpočetního času, dospěl k řešení. Toto řešení splňuje okrajové podmínky s normou odchylky o velikosti 5e-6. V následující tabulce můžete nalézt porovnání hodnoty kritéria získaného pomocí gradientní metody 1. řádu a kritéria získaného analytickou cestou.

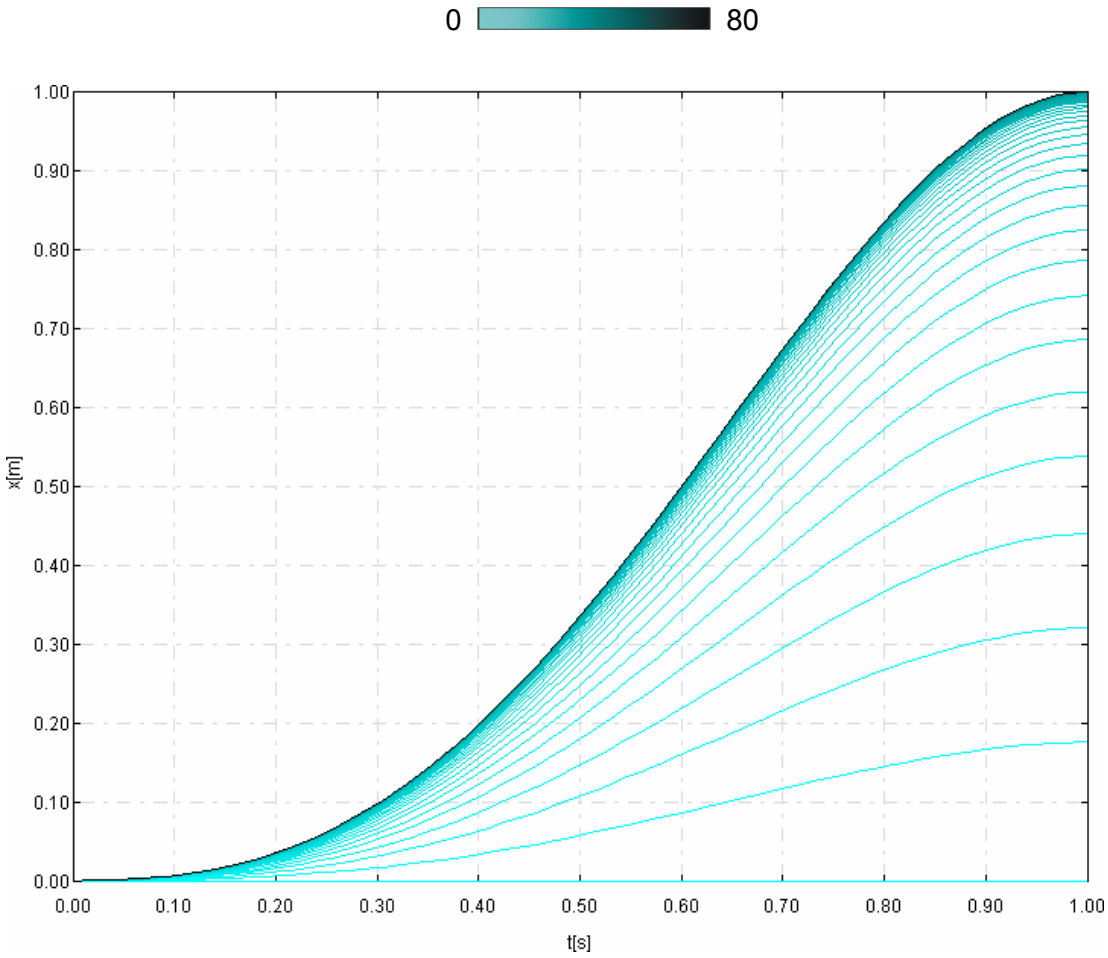
metoda	J
gradientní 1. řádu	45.630419
analytická	45.627385

Tabulka 3.1: Porovnání hodnoty kritéria gradientní metody 1.řádu.

Z tabulky je zřejmé, že gradientní metoda dosáhla velmi dobrého řešení, které se od řešení získaného analytickou cestou liší pouze o 0,07 promile. Tato odchylka je způsobena numerickou chybou výpočtu a délkou simulačního kroku.

Na následujícím obrázku si popíšeme vlastnosti chování gradientní metody při hledání extrému. Jedná se o graf znázorňující průběhy souřadnic polohy *x* (stavové proměnné x_1) v jednotlivých iteracích výpočtu. Iterace jsou zobrazeny barevným přechodem - azurová barva odpovídá iteraci 0 (počáteční odhad) až černá barva odpovídá iteraci 80. Je vidět, že k největším úpravám dochází v prvních třinácti iteracích. Od čtrnácté iterace můžeme pozorovat zhuštění křivek. Křivky generované během posledních 40 iteracích se již téměř překrývají. To vyplývá z povahy této metody. Gradientní metody 1.řádu se projevují

největší změnou vektoru řízení v několika prvních iteracích výpočtu, ale poté již konvergují velmi pomalu. Kompletní vývoj průběhů polohy *x*, rychlosti *v*, zrychlení *a* a řízení *u* hmotného bodu v jednotlivých iteracích lze nalézt v příloze 1A.



Obr. 3.2: Graf průběhu souřadnic polohy *x* v jednotlivých iteracích.

Pro názornější demonstraci konvergence této metody k řešení je v následující tabulce uveden průběh výpočtu v jednotlivých iteracích. V prvním sloupečku jsou obsažena čísla iterací *n*, v následujícím sloupečku jsou normy odchylek dosaženého stavového vektoru, v čase t_1, od požadovaného stavového vektoru (definovaného v okrajových podmínkách). Poslední sloupeček vyjadřuje změnu gradientu *dJ* oproti předchozí iteraci.

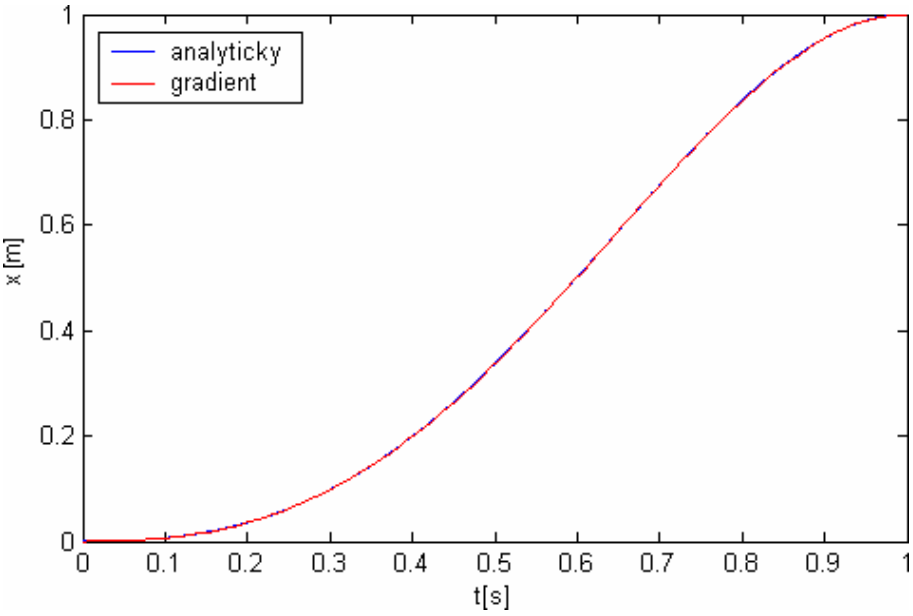
n	norma	dJ
0	1.000000	x
1	0.824747	-3.197918E+006
2	0.680039	-2.177547E+006
3	0.560839	-1.479126E+006
4	0.462453	-1.006770E+006
5	0.381382	-6.841017E+005
6	0.314486	-4.655043E+005
7	0.259349	-3.163874E+005
8	0.213862	-2.152472E+005
9	0.176365	-1.463205E+005
10	0.145435	-9.953244E+004

n	norma	dJ
70	0.000007	-1.924646E-005
71	0.000006	-1.308717E-005
72	0.000006	-8.898840E-006
73	0.000006	-6.050798E-006
74	0.000005	-4.114164E-006
75	0.000005	-2.797294E-006
76	0.000005	-1.901864E-006
77	0.000005	-1.293012E-006
78	0.000005	-8.790289E-007
79	0.000005	-5.975537E-007
80	0.000005	-4.061793E-007

Tabulka 3.2: Průběh výpočtu gradientní metody 1.řádu v jednotlivých iteracích.

Z této tabulky je jasně viditelná funkce metody. Metoda upravuje v jednotlivých iteracích vektor řízení takovým způsobem, že funkce kritéria *J* má záporný gradientní spád. Velikost *dJ* se spolu s rostoucím počtem iterací zmenšuje a při dosažení dostatečně malého *dJ* je algoritmus ukončen.

Na závěr graficky porovnáme průběhy polohy *x* získané touto metodou s průběhy polohy získanými analytickou cestou.



Obr. 3.3: Porovnání průběhů polohy *x* získaných pomocí gradientní a analytické metody.

Z obrázku je patrné, že se oba průběhy překrývají. Můžeme tedy říci, že naše realizace gradientní metody 1. řádu je funkční, konverguje k řešení a toto řešení se od výsledků získaných analytickou cestou liší jen velmi málo.

3.4. Řešení pomocí metody nelineární střelby

Počet bodů simulace byl opět nastaven na hodnotu 101, což odpovídá simulačnímu kroku o velikosti 0.01s. Jako počáteční odhad vektoru λ jsme zvolili hodnotu $\lambda = (-1,-1)$. Algoritmus je schopný najít řešení již po jedné iteraci, ale pro lepší názornost byl nastaven počet iterací na hodnotu tři. Po spuštění výpočtu, algoritmus v jednotlivých iteracích upravoval vektor λ a po třech iteracích a 0.3s výpočetního času dospěl k řešení. Toto řešení splňuje okrajové podmínky s normou odchylky o velikosti 5e-6. V následující tabulce můžete vidět porovnání hodnoty kritéria získaného pomocí metody nelineární střelby a kritéria získaného analytickou cestou:

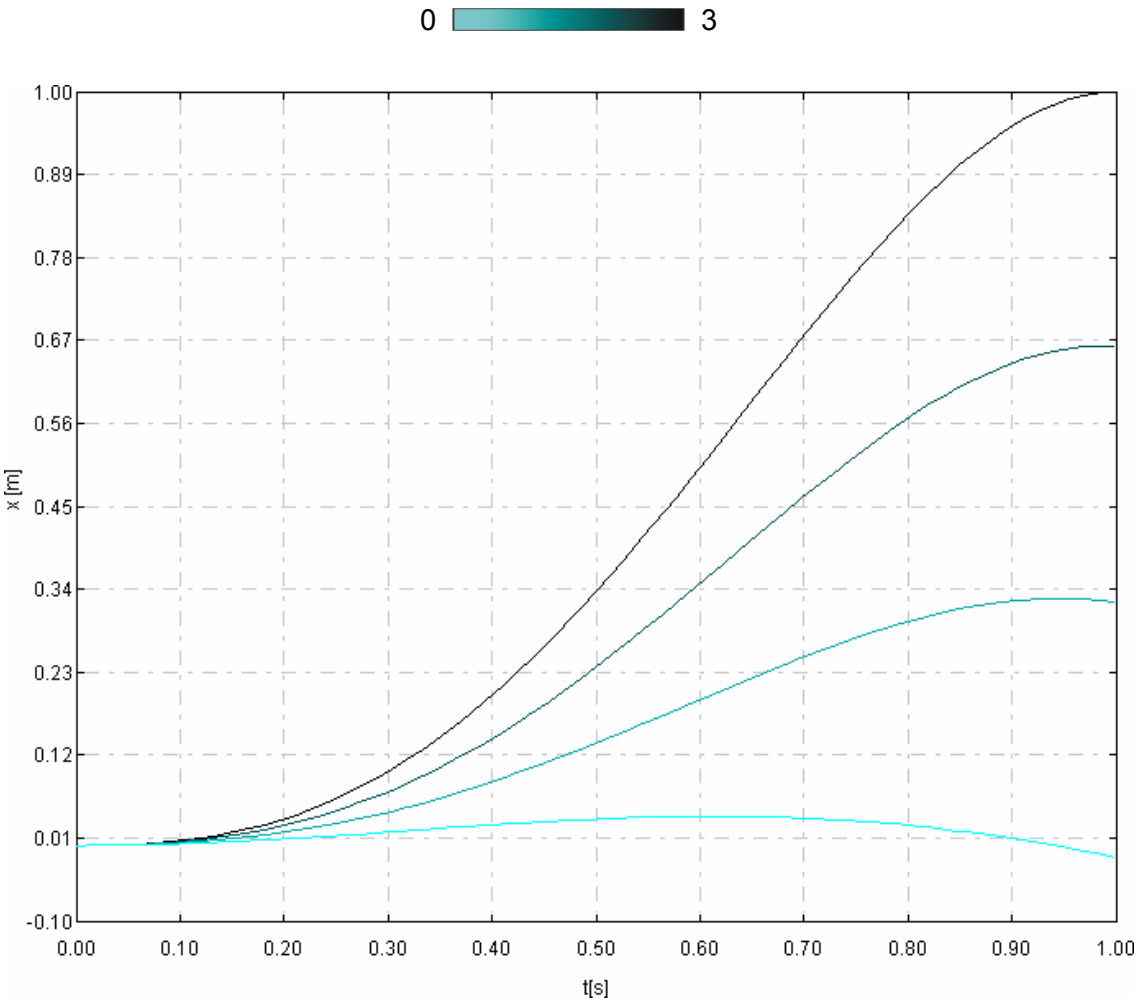
metoda	J
nelineární střelba	45.631163
analytická	45.627385

Tabulka 3.3: Porovnání hodnoty kritéria metody nelineární střelby.

Z tabulky je zřejmé, že metoda nelineární střelby dosáhla o něco horšího řešení než gradientní metoda 1. řádu. To je způsobeno charakterem metody, kde se s klesající délkou simulačního kroku snižuje hodnota funkce kritéria. Přesto je řešení velmi dobré a liší se od analytické pouze o 0.09 promile.

Na následujícím obrázku si popíšeme vlastnosti chování metody nelineární střelby při hledání extrému. Jedná se o graf znázorňující průběhy souřadnic polohy x (stavové proměnné x_1) v jednotlivých iteracích výpočtu. Iterace jsou zobrazeny barevným přechodem - azurová barva odpovídá iteraci 0 až černá barva odpovídá iteraci 3. Je dobře viditelné, že tato metoda konverguje k řešení rychle a s konstantní délkou kroku. Tato vlastnost je velkou výhodou. Podle výsledků řešení na této jednoduché úloze by jsme snadno mohli říci, že je lepší tuto metodu používat spíše než gradientní metodu 1. řádu. Tato metoda má však také své nevýhody a těmi je problém nalezení dobrého počátečního odhadu, zejména u silně nelineárních a složitých systémů. V tomto případě jsme vystačili

pouze s odhadem λ = *(-1,-1)*. Kompletní vývoj polohy *x*, rychlosti *v*, zrychlení *a* a řízení *u* hmotného bodu v jednotlivých iteracích lze nalézt v příloze 1B.



Obr. 3.4: Graf průběhů souřadnic polohy *x* v jednotlivých iteracích.

Stejně jako u gradientní metody, pro demonstraci konvergence této metody k řešení, je v následující tabulce uveden průběhu výpočtu v jednotlivých iteracích. V prvním sloupečku jsou čísla iterací *n*, v následujícím sloupečku jsou normy odchylek současného stavového vektoru v čase t_1 od požadovaného stavového vektoru (definovaného v okrajových podmínkách) . Poslední sloupeček obsahuje hodnotu vektoru λ v iteraci *n*.

n	norma	λ
0	1.054668	(-1.00, -1.00)
1	0.703112	(9.85, -1.26)
2	0.351556	(20.71, -1.52)
3	0.000001	(31.56, -1.78)

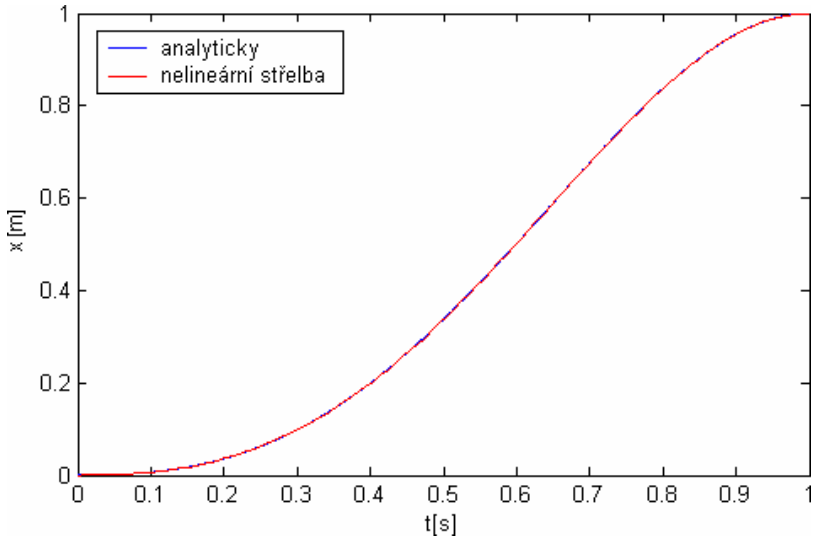
Tabulka 3.4: Průběh výpočtu metody nelineární střelby v jednotlivých iteracích.

Vidíme, že se počáteční norma odchylky stavového vektoru v čase t_1 v jednotlivých iteracích snižuje a to přibližně o konstantní hodnotu. Stejně tak se v jednotlivých iteracích vyvíjí hodnota vektoru λ takovým způsobem, aby v poslední iteraci byly splněny okrajové podmínky.

Stejným způsobem výpočet probíhá i když zvolíme odlišné počáteční hodnoty vektoru λ, např. $\lambda = (100,100)$. Průběh výpočtu můžete nalézt v následující tabulce.

n	norma	λ
0	28.446409	(100.00, 100.00)
1	18.964273	(77.19, 66.07)
2	9.482136	(54.37, 32.15)
3	0.000001	(31.56, -1.78)

Tabulka 3.5: Průběh výpočtu metody nelineární střelby v jednotlivých iteracích.



Obr. 3.5: Porovnání průběhů polohy *x* získaných pomocí nelineární střelby a analyticky.

Na závěr je uvedeno grafické porovnání průběhu polohy *x* získané touto metodou s průběhy poloh získaných analytickou cestou. Výsledky jsou zobrazeny na obr. 3.5. Celkové porovnání optimálních průběhů polohy *x*, rychlosti *v*, zrychlení *a* a řízení *u* hmotného bodu získaných pomocí gradientní metody 1. řádu (na grafu červenou barvou), metody nelineární střelby (na grafu zelenou barvou) a jejich analytické řešení (na grafu modrou barvou) lze nalézt v příloze 1C.

Z obrázku je patrné, že se oba průběhy překrývají. Můžeme tedy říci, že naše realizace metody nelineární střelby je funkční. Metoda konverguje k řešení velice rychle a pro takto jednoduchý systém je možné volit počáteční odhad vektoru λ z poměrně širokého rozsahu hodnot, aniž by výpočet selhal. Vyzkoušeli jsme dva různé počáteční odhady a z obou metoda konvergovala ke stejnému řešení, které se od výsledků získaných analytickou cestou liší jen velmi málo.

Porovnání výsledků získaných pomocí obou metod s analytickým řešením můžete nalézt v následující tabulce.

metoda	J	t[s]	norma
gradientní 1. řádu	45.630419	0.85	0.000005
nelineární střelba	45.631163	0.3	0.000001
analytická	45.627385	x	0.000000

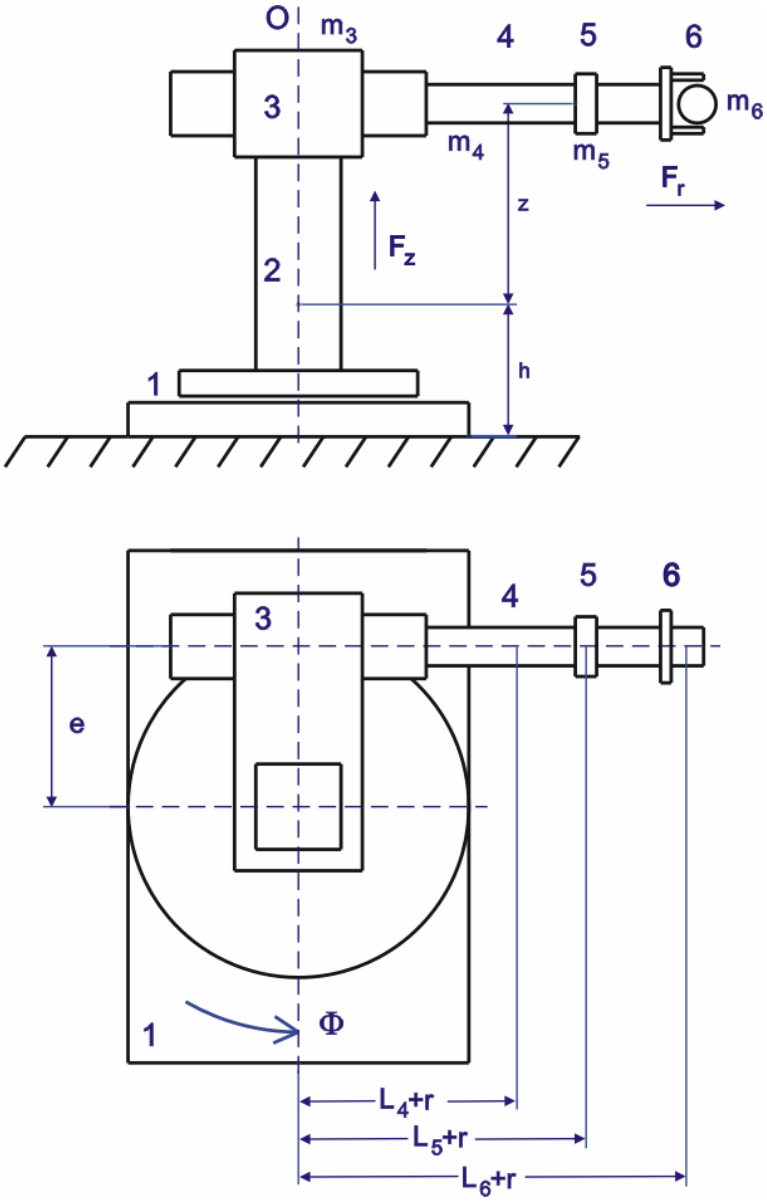
Tabulka 3.6: Celkové porovnání obou metod s analytickým řešením.

Obě metody konvergovaly ke shodnému řešení, které se blíží řešení získanému analytickou cestou pomocí variačního počtu. Přesného řešení ovšem nelze dosáhnout a to zejména proto, že tyto metody jsou numerického charakteru a v průběhu výpočtu dochází ke vzniku chyb. Přesnost výpočtu závisí také na volbě kroku simulace. Čím menší krok simulace (větší rozlišení) zvolíme, tím větší přesnosti dosáhneme. Metoda nelineární střelby dosáhla řešení za výrazně kratší čas než gradientní metoda 1. řádu a také vykázala velice rychlou konvergenci k řešení. Pro takto jednoduchý případ se tedy tato metoda projevila výrazně lépe a to zejména vzhledem k faktu, že jsme nemuseli komplikovaně hledat hodnotu vhodného počátečního odhadu.

4. Řešení základních struktur robotů

4.1. Robot pracující v cylindrickém souřadném systému

Jedná se o velmi rozšířený typ robotu, jehož schéma naleznete na obrázku 4.1. Budeme uvažovat pouze polohovací mechanismus se třemi stupni volnosti, tedy rotaci kolem svislé osy *O*, vertikální posun a horizontální výsun ramene.



Obr. 4.1: Schéma robotu pracujícího v cylindrickém souřadném systému.

Nejprve vysvětleme pohyby, které je robot schopen vykonávat a tyto pohyby popišme pomocí proměnných. Druhý člen robotu se vzhledem ke členu 1 pohybuje rotačně kolem svislé osy *O*, úhel natočení popisujeme pomocí proměnné Φ. Člen 3 se pohybuje posuvně vzhledem ke členu 2 (posun nahoru a dolů), popisujeme pomocí vertikálního posunu *z*. Člen 4 se vzhledem ke členu 3 pohybuje také posuvně, popisujeme pomocí výsunu ramene *r*.

Popišme význam jednotlivých parametrů tohoto robotu. Hmotný bod m_5 je redukovanou hmotností zápěstí, m_6 je redukovanou hmotností uchopeného břemene (nebo také technologické hlavice). Hmotný body m_3 je redukovanou hmotností členu 3 (horní část ramene) a hmotný bod m_4 je redukovanou hmotností členu 4 (výsuvné části ramene). Konstanty I_2, I_3, I_4 jsou po řadě momenty setrvačnosti členů 2 a 3 vzhledem k ose *O* a členu 4 k těžišti T_4. Konstanty L_4, L_5, L_6 jsou po řadě polohy těžišť členů 4, 5, 6 při zasunutém rameni robotu, *h* je minimální možná poloha ramene ve směru osy *z* a *e* je posunutí výsuvné části ramene od osy rotace.

Číselné údaje pro konkrétní úlohu:

$$m_3 = 20kg,\ m_4 = 10kg,\ m_5 = 3kg,\ m_6 = 4kg,\ I_2 = 0.2kg*m^2,$$

$$I_3 = 11kg*m^2,\ I_4 = 26.4kg*m^2,\ L_4 = -0.3m,\ L_5 = 0.4m,\ L_6 = 0.6m, \qquad (4.1.1)$$

$$h = 1m,\ e = 0.15m,\ g = 9.81m*s^{-2}.$$

Pro zkrácení a zjednodušení tvaru pohybových rovnic je výhodné zavést následující parametry:

$$A = \sum_{i=4}^{6} m_i\ ,\ B = \sum_{i=4}^{6} L_i m_i\ ,\ C = \sum_{i=2}^{4} I_i + \sum_{i=4}^{6} m_i \left(e^2 + L_i^{\,2}\right),\ D = \sum_{i=3}^{6} m_i\ . \qquad (4.1.2)$$

Pohybové rovnice robotu pak mají tvar:

$$F_r = A\ddot{r} - \dot{\Phi}^2 \left(Ar + B\right) - eA\ddot{\Phi}\ ,$$

$$M_\Phi = \ddot{\Phi}\left(Ax^2 + 2Bx + C\right) + 2\dot{\Phi}\dot{x}\left(Ax + B\right) - eA\ddot{x}\ ,$$

$$F_z = D\ddot{z} + gD\ . \qquad (4.1.3)$$

Pro stavový popis ramene robotu definujeme 6 stavových proměnných (tři pro pozici a tři pro rychlost):

$$x_1 = z,\ x_2 = \dot{z},\ x_3 = \Phi,\ x_4 = \dot{\Phi},\ x_5 = r,\ x_6 = \dot{r}. \tag{4.1.4}$$

Nechť F_z je hnací síla v ose z, M_Φ je hnací moment v ose Φ a F_r je hnací síla v ose r. Zavedeme tři zobecněné proměnné pro řízení ramene robotu:

$$u_1 = F_z \big/ \left(k_{m1} i_{m1}\right),\ u_2 = M_\Phi \big/ \left(k_{m2} i_{m2}\right),\ u_3 = F_r \big/ \left(k_{m3} i_{m3}\right), \tag{4.1.5}$$

kde i_m značí zjednodušený převodový poměr mezi zobecněnou silou (resp. momentem) a pohonem, k_m představuje ztrátový poměr. V této úloze volíme i_m = 100, k_m = 0.68 u všech pohonů.

Nyní jsme již zobecnili všechny potřebné proměnné v dynamickém popisu robotu (4.1.3) a můžeme sestavit jeho stavový popis:

$$\begin{aligned}
&f_1 : \dot{x}_1 = x_2 \\
&f_2 : \dot{x}_2 = F_z / D - g \\
&f_3 : \dot{x}_3 = x_4 \\
&f_4 : \dot{x}_4 = \left(M_\Phi - 2x_4 x_6 \left(Ax_5 + B\right) + eA\dot{x}_6\right) / \left(Ax_5^2 + 2Bx_5 + C\right) \\
&f_5 : \dot{x}_5 = x_6 \\
&f_6 : \dot{x}_6 = \left(F_r + x_4^2 \left(Ax_5 + B\right) + eA\dot{x}_4\right) / A.
\end{aligned} \tag{4.1.6}$$

Jak vidíme, je zde jeden problém. Ve stavovém popisu se vyskytuje algebraická smyčka. Úlohu s algebraickou smyčkou není možné řešit přímo. Aby se průběh výpočtu algoritmu zbytečně nezpomalil, bylo zvoleno řešení analytické. Do výrazu f_4 byl dosazen výraz f_6 a analogicky se do výrazu f_6 dosadil výraz f_4. Po úpravě jsme tedy dospěli k novému stavovému popisu, který již algebraickou smyčku neobsahuje:

$$\begin{aligned}
&f_1 : \dot{x}_1 = x_2 \\
&f_2 : \dot{x}_2 = F_z / D - g \\
&f_3 : \dot{x}_3 = x_4
\end{aligned} \tag{4.1.7}$$

$$f_4 : \dot{x}_4 = \left(M_\Phi - 2x_4x_6\left(Ax_5 + B\right) + eF_r + ex_4^2\left(Ax_5 + B\right)\right)\Big/\left(Ax_5^2 + 2Bx_5 + C - Ae^2\right)$$
$$f_5 : \dot{x}_5 = x_6$$
$$f_6 : \dot{x}_6 = \left[\left(F_r + x_4^2\left(Ax_5 + B\right)\right)\cdot\left(Ax_5^2 + 2Bx_5 + C\right) + eA\left(M_\Phi - 2x_4x_6\left(Ax_5 + B\right)\right)\right]\Big/$$
$$\Big/\left(A^2x_5^2 + 2BAx_5 + AC - A^2e^2\right). \qquad (4.1.7)$$

Je vidět, že se stavový popis řešením algebraické smyčky poněkud zkomplikoval a přibylo nelinearit.

Dále zavedeme funkci kritéria optimalizace, jako integrál součtu kvadrátů hodnot řízení:

$$J = \int_{t_0}^{t_1} E\left(t, x\left(t\right), u\left(t\right)\right)dt \text{, kde } E\left(x,u,t\right) = u_1^2 + u_2^2 + u_3^2. \qquad (4.1.8)$$

Robot obsahuje jeden rotační a dva posuvné členy. Navíc posuvný pohyb ramene robotu (členu 3 vůči členu 2) je kolmý ke zbývajícím dvou pohybům, proto je dynamicky neovlivní. Systém tohoto robotu je výrazně jednodušší než systémy ostatních struktur robotů a právě proto je velmi vhodný pro odzkoušení funkce obou metod.

Nyní zavedeme hodnotu stavového vektoru v čase *t* jako:

$$x\left(t\right) = \left[x_1\left(t\right) \quad \cdots \quad x_n\left(t\right)\right]. \qquad (4.1.9)$$

Formulujeme okrajové podmínky pro řešení tohoto systému ve tvaru: počáteční čas $t_0 = 0s$, koncový čas $t_1 = 2s$, hodnota stavového vektoru v počátečním čase a koncovém čase:

$$x\left(t_0\right) = \left[0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0\right],\ x\left(t_1\right) = \left[1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0\right]. \qquad (4.1.10)$$

Což odpovídá v čase t_0 výsunu ramene $r = 0m$, vertikálního posunu $z = 0m$, úhlu natočení Φ = 0rad a rychlosti všech členů rovnou nule. V čase t_1 pak výsunu ramene $r = 1m$, vertikálního posunu $z = 1m$, úhlu natočení $\Phi = 1rad$ ($\Phi \approx 57.3°$) a rychlosti všech členů rovnou nule.

Tento typ robotu jsme nejprve řešili pomoci gradientní metody 1. řádu. Jako odhad počátečního řízení byly zvoleny konstantní funkce o nulové hodnotě: $u_1(t) = 0$, $u_2(t) = 0$, $u_3(t) = 0$. Počáteční odhad konstanty α byl nastaven na hodnotu $\alpha = 1e\text{-}6$ a při výpočtu

jsme využili její automatický odhad. Hodnotu konstanty penalizace splnění okrajových podmínek jsme nastavili na hodnotu *K = 1e7*. Počet bodů simulace byl zvolen o hodnotě 52. Algoritmus dospěl k řešení po 500 iteracích a 4.1 sekundách výpočetního času. Rychlost výpočtu je velmi vysoká a to i vzhledem ke špatnému počátečnímu odhadu řízení. To jsme ovšem předpokládali, protože robot pracující v cylindrickém souřadném systému je pro řešení velice jednoduchý. Velikost kritéria řešení jsme získali o velikosti *J = 58.08218*. Norma odchylky výsledného řešení je 0.000410. Z průběhu výpočtu opět vidíme charakter gradientní metody 1. řádu. V prvních 120 iteracích dochází k největším změnám ve vývoji řešení a ve zbylých iteracích dochází již jen k velmi malým změnám a metoda velice pomalu konverguje k řešení.

Dále byla tato úloha řešena pomocí metody nelineární střelby. Počet bodů simulace byl opět nastaven na hodnotu 52 a počet iterací na hodnotu 10. Jako počáteční odhad vektoru λ jsme zvolili $\lambda = (-1, -1, -1, -1, -1, -1)$, tedy stejně jako u gradientní metody, velmi špatný počáteční odhad. To ovšem u takto jednoduchého systému nevadí. Algoritmus našel řešení již po 10 iteracích a 2.2 sekundách výpočetního času. Kritérium získaného řešení je *J = 58.083911* a norma odchylky od okrajových podmínek 1e-6. Je vidět, že tato metoda našla řešení přibližně 2x rychleji než gradientní metoda. Toto řešení má mírně horší kritérium, ale na druhou stranu má lepší normu odchylky od splnění okrajových podmínek. Také je viditelné, že výpočetní náročnost jedné iterace metody nelineární střelby je výrazně vyšší než u gradientní metody.

Výsledky řešení robotu pracujícího v cylindrickém souřadném systému lze nalézt v příloze 2A, kde jsou obsaženy průběhy poloh (na levé straně) a průběhy rychlostí (na pravé straně). V příloze 2B jsou obsaženy průběhy zrychlení (na levé straně) a průběhy řízení (na pravé straně). Modrou barvou jsou znázorněny výsledky získané pomocí gradientní metody 1.řádu, červenou barvou pak výsledky získané pomocí metody nelineární střelby. Z výsledků vidíme, že řešení získané pomocí obou metod se přibližně překrývají.

Výsledky řešení obou metod shrňme do následující tabulky. V prvním sloupečku je název metody, ve druhém doba výpočtu v sekundách. Třetí sloupec obsahuje hodnotu kritéria *J* získanou pomocí dané metody, čtvrtý sloupec obsahuje počet iterací *n* nutných k dosažení řešení. Pátý sloupec obsahuje normu odchylky řešení od splnění okrajových

podmínek (4.1.10) a poslední sloupec obsahuje rozlišení dané metody, tedy počet bodů, ve kterých je systém simulován.

metoda	t[s]	J	n	norma	rozlišení
gradientní 1. řádu	4.1	58.082180	500	0.000410	52
nelineární střelby	2.2	58.083911	10	0.000001	52

Tabulka 4.1: Výsledky řešení robotu pracujícího v cylindrickém souřadném systému.

Dále jsme výsledky naší gradientní metody realizované v C++ porovnali s výsledky získanými gradientní metodou 1. řádu realizovanou v prostředí MATLAB. Porovnání je zobrazeno v následující tabulce.

metoda	t[s]	J	n	norma	rozlišení
gradientní C++	4.1	58.082180	500	0.000410	52
gradientní MATLAB	37.0	58.179500	610	0.000450	61

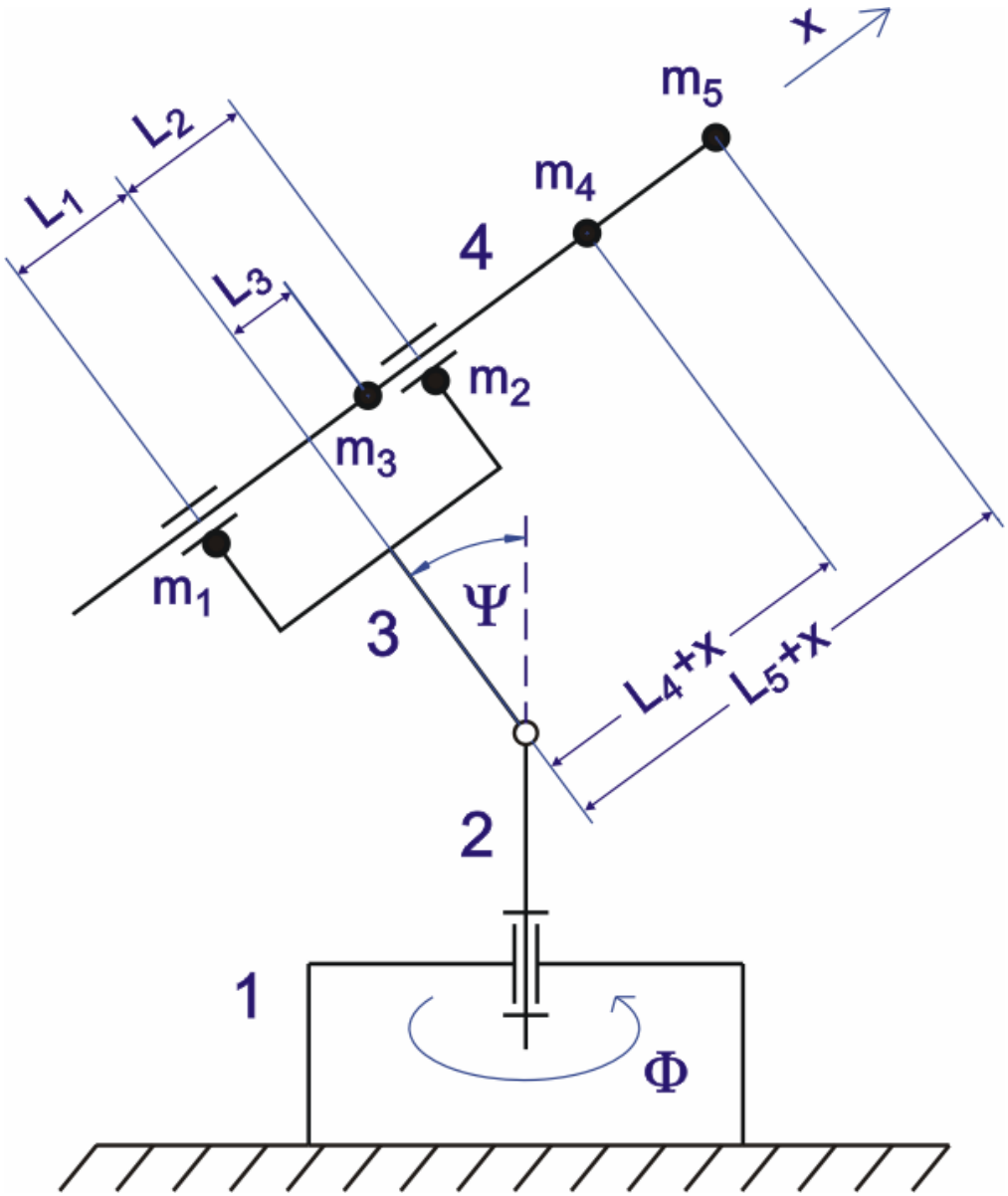
Tabulka 4.2: Porovnání gradientní metody 1. řádu realizované v C++ a MATLAB.

Porovnání výsledků obou metod získaných při řešení robotu pracujícího v cylindrickém souřadném systému lze nalézt v příloze 2C, kde jsou obsaženy průběhy poloh (na levé straně) a průběhy rychlostí (na pravé straně). V příloze 2D jsou zobrazeny průběhy zrychlení (na levé straně) a průběhy řízení (na pravé straně). Modrou barvou jsou znázorněny výsledky získané pomocí gradientní metody 1.řádu realizované v jazyce C++, červenou barvou pak výsledky získané pomocí gradientní metody 1.řádu realizované v prostředí MATLAB.

Z porovnání výsledku můžeme tvrdit, že v tomto případě naše metoda realizovaná v jazyce C++ dospěla k řešení zhruba 9x rychleji a to dokonce s vyšší přesností splnění okrajových podmínek a lepší hodnotou kritéria než metoda realizovaná v prostředí MATLAB.

4.2. Robot pracující ve sférickém souřadném systému

Kinematické schéma robotu se sférickým pracovním prostorem je zobrazeno na obr. 4.2. Orientační ústrojí chapadla je zanedbáno, uvažujeme pouze polohovací systém, který má 3 stupně volnosti a to rotaci kolem svislé osy, kývavý pohyb kolem vodorovné osy a posuvný pohyb ve směru osy x.



Obr. 4.2: Kinematické schéma robotu pracujícího ve sférickém souřadném systému.

Nejprve vysvětleme pohyby, které je robot schopen vykonávat a tyto pohyby popišme pomocí proměnných. Člen robotu 2 se vzhledem ke členu 1 pohybuje rotačně kolem svislé osy - úhel natočení popisujeme pomocí proměnné Φ. Člen 3 se pohybuje rotačně vzhledem ke členu 2 - úhel natočení popisujeme pomocí proměnné Ψ (kývavý pohyb). Člen 4 se vzhledem ke členu 3 pohybuje posuvně (výsun ramene) - popisujeme pomocí proměnné *x*.

Popišme význam jednotlivých parametrů tohoto robotu. Hmotnost členu 2 a jeho moment setrvačnosti k svislé ose rotace zanedbáváme. Hmotný bod m_4 je zobecněnou hmotností zápěstí při zasunutém rameni ve vzdálenosti L_4 od osy rotace členu 3 vůči členu 2. Hmotný bod m_5 je zobecněnou hmotností uchopeného břemene (nebo také technologické hlavice) při zasunutém rameni ve vzdálenosti L_5 od osy rotace členu 3 vůči členu 2. Dále pak hmotné body m_1, m_2, m_3 jsou redukované hmotnosti jednotlivých částí ramene robotu, které spolu s jejich vzdálenostmi L_1, L_2, L_3 (polohy hmotných bodů při zasunutém rameni od osy rotace členu 3 vůči členu 2) nahrazují silové působení jednotlivých částí ramene robotu.

Číselné údaje pro konkrétní úlohu:

$$m_1 = 40\text{kg},\ m_2 = 23\text{kg},\ m_3 = 93\text{kg},\ m_4 = 25\text{kg},\ m_5 = 20\text{kg},$$

$$L_1 = -0.75\text{m},\ L_2 = 0.62\text{m},\ L_3 = -0.72\text{m},\ L_4 = 1.22\text{m},\ L_5 = 1.25\text{m}, \qquad (4.2.1)$$

$$e = 0.3\text{m},\ g = 9.81\text{ms}^{-2}.$$

Pro zkrácení a zjednodušení tvaru pohybových rovnic je výhodné zavést následující parametry:

$$C_1 = \sum_{i=3}^{5} m_i\ ,\ C_2 = \sum_{i=3}^{5} L_i m_i\ ,\ C_3 = \sum_{i=1}^{5} L_i^2 m_i\ ,\ C_4 = \sum_{i=1}^{5} m_i\ ,\ C_5 = \sum_{i=1}^{5} L_i m_i\ . \qquad (4.2.2)$$

Pohybové rovnice robotu pak mají tvar:

$$\ddot{x}C_1 = F_x - gC_1 \sin\Psi + \dot{\Psi}^2\left(C_1 x + C_2\right) + \Phi^2\left[\left(C_1 x + C_2\right)\cdot\cos^2\Psi - 0.5C_1 e\sin 2\Psi\right], \qquad (4.2.3)$$

$$\ddot{\Psi}\left(C_1x^2+2C_2x+C_3+C_4e^2\right)=M_\Psi-g\left[\left(C_1x+C_5\right)\cos\Psi-C_4e\sin\Psi\right]-$$
$$-\Phi^2\left[0.5\left(C_1x^2+2C_2x+C_3-C_4e^2\right)\sin2\Psi+e\left(C_1x+C_5\right)\cos2\Psi\right]-$$
$$-2\dot{\Psi}\dot{x}\left(C_1x+C_2\right),$$

$$\ddot{\Phi}\left[\left(C_1x^2+2C_2x+C_3\right)\cos^2\Psi-e\left(C_1x+C_5\right)\sin2\Psi+C_4e^2\sin^2\Psi\right]=$$
$$=M_\Phi-\dot{\Phi}\dot{x}\left[2\left(C_1x+C_2\right)\cos^2\Psi-C_1e\sin2\Psi\right]+$$
$$+\dot{\Psi}\dot{\Phi}\left[\left(C_1x^2+2C_2x+C_3-C_4e^2\right)\sin2\Psi+2e\left(C_1x+C_5\right)\cos2\Psi\right]. \qquad (4.2.3)$$

Pro stavový popis ramene robotu definujeme 6 stavových proměnných (tři pro pozici a tři pro rychlost):

$$x_1=x,\ x_2=\dot{x},\ x_3=\Psi,\ x_4=\dot{\Psi},\ x_5=\Phi,\ x_6=\dot{\Phi}. \qquad (4.2.4)$$

Nechť F_x je hnací síla v ose x, M_Ψ je hnací moment v ose Ψ a M_Φ je hnací moment v ose Φ. Zavedeme tři zobecněné proměnné pro řízení ramene robotu:

$$u_1=F_x/\left(k_{m1}i_{m1}\right),\ u_2=M_\Psi/\left(k_{m2}i_{m2}\right),\ u_3=M_\Phi/\left(k_{m3}i_{m3}\right), \qquad (4.2.5)$$

kde i_m značí zjednodušený převodový poměr mezi zobecněnou silou (resp. momentem) a pohonem, k_m představuje ztrátový poměr. V této úloze volíme i_m = 100, k_m = 0.68 u všech pohonů.

Nyní jsou již zobecněny všechny potřebné proměnné v dynamickém popisu robotu (4.2.3) a lze sestavit jeho stavový popis:

$$f_1:\dot{x}_1=x_2$$
$$f_2:\dot{x}_2=\left\{F_x-C_1g\sin(x_3)+x_4^2\left(C_1x_1+C_2\right)+\right.$$
$$\left.+x_6^2\left[\left(C_1x_1+C_2\right)\cdot\cos^2x_3-0.5C_1e\sin2x_3\right]\right\}/C_1$$
$$f_3:\dot{x}_3=x_4 \qquad (4.2.6)$$

$$f_4 : \dot{x}_4 = \left\{ M_\Psi - g\left[(C_1 x_1 + C5)\cos x_3 - C_4 e \sin x_3 \right] - \right.$$

$$-x_6^2 \left[0.5\left(C_1 x_1^2 + 2C_2 x_1 + C_3 - C_4 e^2 \right)\sin 2x_3 + e\left(C_1 x_1 + C_5 \right)\cos 2x_3 \right] -$$

$$\left. -2x_4 x_2 \left(C_1 x_1 + C_2 \right) \right\} \Big/ \left(C_1 x_1^2 + 2C_2 x_1 + C_3 + C_4 e^2 \right)$$

$$f_5 : \dot{x}_5 = x_6$$

$$f_6 : \dot{x}_6 = \left\{ M_\Phi - x_6 x_2 \left[2\left(C_1 x_1 + C_2 \right)\cos^2 x_3 - C_1 e \sin 2x_3 \right] + \right.$$

$$+ x_6 x_4 \left[\left(C_1 x_1^2 + 2C_2 x_1 + C_3 - C_4 e^2 \right)\sin 2x_3 + 2e\left(C_1 x_1 + C_5 \right)\cos 2x_3 \right] \Big\} \Big/$$

$$\Big/ \left[\left(C_1 x_1^2 + 2C_2 x_1 + C_3 \right)\cos^2 x_3 - e\left(C_1 x_1 + C_5 \right)\sin 2x_3 + C_4 e^2 \sin^2 x_3 \right]. \qquad (4.2.6)$$

Ze stavového popisu si můžeme všimnout jedné zásadní výhody a to, že se v něm nevyskytují žádné algebraické smyčky. Nemusíme je tedy řešit a tím dále komplikovat stavový popis tohoto robotu.

Dále zavedeme funkci kritéria optimalizace, jako integrál součtu kvadrátů hodnot řízení:

$$J = \int_{t_0}^{t_1} E\left(x(t), u(t), t \right) dt \text{ , kde } E(x,u,t) = u_1^2 + u_2^2 + u_3^2 \qquad (4.2.7)$$

Robot obsahuje dva rotační a pouze jeden posuvný člen. Dále žádný z jeho pohybů není kolmý k pohybům zbývajícím. Právě proto můžeme ve stavovém popisu pozorovat vzájemnou vazbu všech stavových proměnných. Tento systém je již velice komplikovaný a silně nelineární. A proto také obtížný na řešení.

Hodnotu stavového vektoru v čase *t* zavedeme stejně jako ve výrazu (4.1.9) a formulujeme okrajové podmínky pro řešení tohoto systému ve tvaru: počáteční čas t_0 = *0s*, koncový čas t_1 = *2s*, hodnota stavového vektoru v počátečním čase a koncovém čase:

$$x\left(t_0 \right) = \left[0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \right],\ x\left(t_1 \right) = \left[1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \right] \qquad (4.2.8)$$

Což odpovídá v čase t_0 výsunu ramene *x = 0m*, kyvu ramene Ψ = *0rad*, úhlu otočení Φ = *0rad* a rychlosti všech členů rovnou nule, v čase t_1 pak výsunu ramene *x = 1m*, kyvu

ramene Ψ = 1rad ($\Psi \approx 57.3°$), úhlu natočení Φ = *1rad* ($\Phi \approx 57.3°$) a rychlosti všech členů rovnou nule.

Tento robot byl nejprve řešen pomoci gradientní metody 1. řádu. Jako odhad počátečního řízení jsme zvolili konstantní funkce, definované ve tvaru: $u_n(t) = k_n$, kde $n = 1, 2, 3$ a k_n je vhodně zvolená konstanta. Počáteční odhad konstanty α byl nastaven na hodnotu $\alpha = 1e\text{-}11$ a při výpočtu byl využit její automatický odhad. Hodnota konstanty penalizace splnění okrajových podmínek byla nastavena na hodnotu $K = 1e11$. Pro počet bodů simulace byla zvolena hodnota 101. Algoritmus dospěl k řešení po 25 000 iteracích a 350 sekundách výpočetního času. Tento typ robotu je již na řešení výrazně složitější. Opět se projevuje největší změna řízení v několika prvních stovkách iterací, kdy se norma odchylky řešení od zadaných okrajových snižuje velmi rychle. Ale při dosažení normy o přibližné hodnotě 2.5 se konvergence algoritmu k řešení silně zpomalí. Po 25 000 iteracích jsme dospěli k řešení s normou odchylky 0.02 a hodnotou kritéria $J = 117.15368$. Pokud by jsme chtěli získat řešení, které je ještě bližší splnění okrajových podmínek, museli by jsme nastavit větší počet iterací. Bohužel je u tohoto typu robotu konvergence k řešení tak pomalá, že pro dosažení řešení o normě odchylky o velikosti 1e-6 potřebujeme zhruba dalších 450 000 iterací a 6 280 sekund výpočetního času.

Dále byla tato úloha řešena pomocí metody nelineární střelby. Počet bodů simulace jsme opět nastavili na hodnotu 101. Tento typ robotu je již složitější a tak jsme s nastavením počátečního odhadu vektoru λ museli již více experimentovat. Po několika pokusech, které skončily selháním výpočtu jsme zvolili hodnotu vektoru λ na $\lambda = (-3e5, 3e4, 3e5, 2e5, 1e3, -1e4)$ a počet iterací na hodnotu 100. Algoritmus pak našel řešení za 192s výpočetního času. Kritérium takto získaného řešení je $J = 117.195345$ a norma odchylky od splnění okrajových podmínek 1e-6. Je patrné, že tato metoda našla řešení přibližně 2x rychleji než gradientní metoda a to s velmi dobrou normou odchylky, kterou by jsme při použití gradientní metody zjišťovali velmi dlouhou dobu. Cenou za takto rychlou konvergenci jsou počáteční problémy s vhodným odhadem vektoru λ.

Také jsme vyzkoušeli, zda je možné získat řešení při zvolení špatného počátečního odhadu a zvýšení počtu iterací. Zvolili jsme tedy jako počáteční odhad vektor $\lambda = (-1, -1, -1, -1, -1, -1)$ a experimentovali s nastavením počtu iterací. Při zvolení počtu iterací o hodnotě 550 byl algoritmus schopen najít řešení. Toto řešení bylo nalezeno za 390s

výpočetního času s normou odchylky od splnění okrajových podmínek o velikosti 1e-6. Při volbě nižšího počtu iterací algoritmus selhal. Toto zjištění je velmi dobré, protože nám říká, že jsme schopni metodou nelineární střelby řešit robotu pracujícího ve sférickém souřadném systému i se špatným počátečním odhadem za dobu přibližně shodnou s gradientní metodou 1. řádu, ale s výrazně lepší normou odchylky.

Výsledky řešení robotu pracujícího ve sférickém souřadném systému lze nalézt v příloze 3A, kde jsou obsaženy průběhy poloh (na levé straně) a průběhy rychlostí (na pravé straně). V příloze 3B jsou obsaženy průběhy zrychlení (na levé straně) a průběhy řízení (na pravé straně). Modrou barvou jsou znázorněny výsledky získané pomocí gradientní metody 1.řádu, červenou barvou pak výsledky získané pomocí nelineární střelby. Z výsledků vidíme, že řešení získané pomocí obou metod se přibližně překrývají.

Řešení obou metod také shrneme v následující tabulce. V prvním sloupečku je název metody, ve druhém doba výpočtu v sekundách. Třetí sloupec obsahuje hodnotu kritéria *J* získaného pomocí dané metody, čtvrtý sloupec obsahuje počet iterací *n* nutných k dosažení řešení. Pátý sloupec obsahuje normu odchylky řešení od splnění okrajových podmínek (4.2.8) a poslední sloupec obsahuje rozlišení dané metody, tedy počet bodů, ve kterých je systém simulován.

metoda	t[s]	J	n	norma	rozlišení
gradientní 1. řádu	350	117.153680	25000	0.020000	101
nelineární střelby	192	117.195345	100	0.000001	101

Tabulka 4.3: Výsledky řešení robotu pracujícího ve sférickém souřadném systému.

Dále jsme výsledky naší gradientní metody realizované v jazyce C++ porovnali s výsledky získanými gradientní metodou 1. řádu realizovanou v prostředí MATLAB. Porovnání je zobrazeno v tabulce 4.4.

Grafické porovnání výsledků obou metod získaných při řešení tohoto robotu lze nalézt v příloze 3C, kde jsou obsaženy průběhy poloh (na levé straně) a průběhy rychlostí (na pravé straně). V příloze 3D jsou obsaženy průběhy zrychlení (na levé straně) a průběhy řízení (na pravé straně). Modrou barvou jsou znázorněny výsledky získané pomocí

gradientní metody 1.řádu realizované v jazyce C++, červenou barvou pak výsledky získané pomocí gradientní metody 1.řádu realizované v prostředí MATLAB.

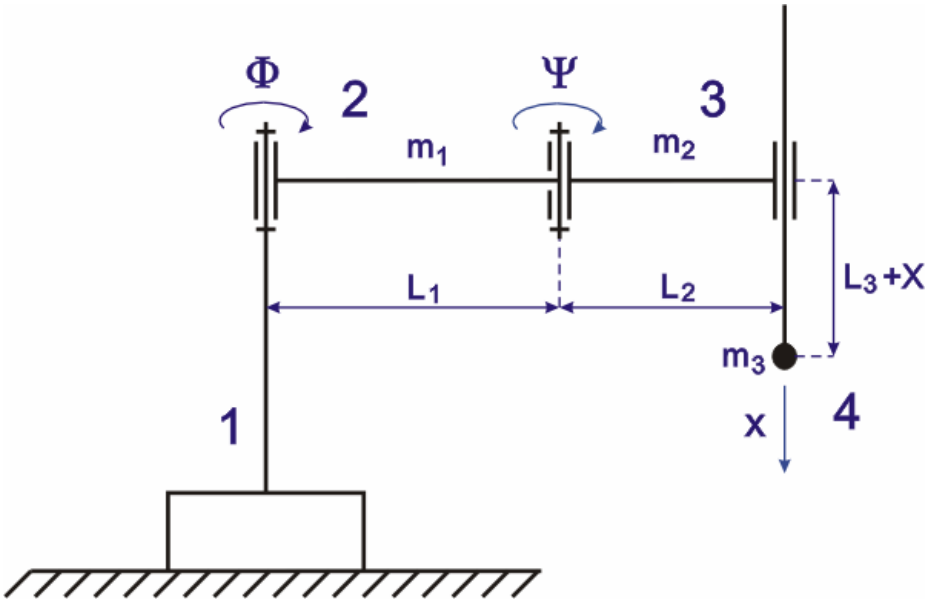
metoda	t[s]	J	n	norma	rozlišení
gradientní C++	350	117.153680	25000	0.020	101
gradientní MATLAB	2312	117.289	34500	0.025	69

Tabulka 4.4: Porovnání gradientní metody 1. řádu realizované v C++ a MATLAB pro robotu pracujícího ve sférickém souřadném systému.

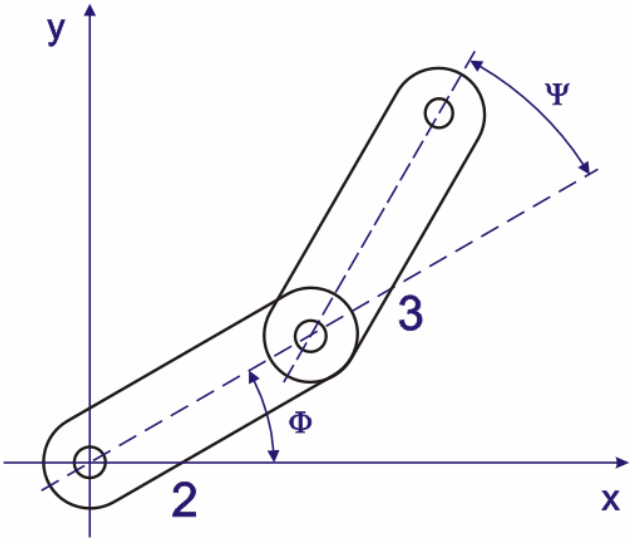
Z porovnání výsledku vidíme, že metoda realizovaná v jazyce C++ dospěla k řešení zhruba 7x rychleji a to i navzdory faktu, že bylo nastaveno vyšším rozlišením. Řešení pomocí metody realizované v jazyce C++ bylo počítáno s vyšší přesností a dokonce i hodnota výsledného kritéria řešení byla nižší než u metody realizované v prostředí MATLAB.

4.3. SCARA robot

Kinematické schéma SCARA robotu je zobrazeno na obr. 4.3. Orientační ústrojí chapadla je zanedbáno, uvažujeme pouze polohovací systém, který má 3 stupně volnosti, tedy rotaci členu 2 kolem svislé osy, rotaci členu 3 kolem svislé osy a posuvný pohyb koncového členu 4 ve směru osy x.



Obr. 4.3: Kinematické schéma SCARA robotu.



Obr. 4.4: Schéma SCARA robotu, pohled shora.

Nejprve vysvětleme pohyby, které je robot schopen vykonávat a tyto pohyby popišme pomocí proměnných. Člen robotu 2 se vzhledem ke členu 1 pohybuje rotačně kolem svislé osy, úhel natočení popisujeme pomocí proměnné Φ. Člen 3 se pohybuje rotačně vzhledem ke členu 2, popisujeme pomocí proměnné Ψ. Člen 4 se vzhledem ke členu 3 pohybuje posuvně (výsun ramene nahoru a dolů), popisujeme pomocí proměnné *x*.

Popišme význam jednotlivých parametrů tohoto robotu. Rozměry L_1, L_2 jsou po řadě délky členů 2, 3 ramene robotu. Hmotnosti m_1, m_2 odpovídají spojitě rozloženým hmotnostem po celé délce členů 2, 3. Hmotnost m_3 představuje redukovanou hmotnost zápěstí robotu spolu s velikostí neseného břemene. Rozměr L_3 odpovídá základní poloze vysunutí členu 4 ramene robotu.

Číselné údaje pro konkrétní úlohu:

$$m_1 = 8kg,\ m_2 = 6kg,\ m_3 = 3kg,$$
$$L_1 = 0.4m,\ L_2 = 0.3m,\ L_3 = 0.2m,\ g = 9.81ms^{-2}. \qquad (4.3.1)$$

Pro zjednodušení tvaru můžeme pohybové rovnice robotu zapsat v maticovém tvaru:

$$M\ddot{\vec{q}} + \vec{v} = \vec{T}, \qquad (4.3.2)$$

kde *M* je matice symetrická podle hlavní diagonály a má tvar:

$$M = \begin{bmatrix} ma_{11} & 0 & 0 \\ 0 & ma_{22} & ma_{23} \\ 0 & ma_{32} & ma_{33} \end{bmatrix}. \qquad (4.3.3)$$

Jednotlivé složky matice M rozepíšeme následovně:

$$\mathrm{ma}_{11} = m_3,\ m_{23} = m_{32},$$
$$\mathrm{ma}_{22} = (m_2 + m_3)L_1^2 + m_3L_2^2 + 2m_3L_1L_2\cos(\Psi), \qquad (4.3.4)$$
$$\mathrm{ma}_{23} = m_3L_2^2 + m_3L_1L_2\cos(\Psi),\ \ m_{33} = m_3L_2^2.$$

Vektor $\vec{v}$ má tvar:

$$v = (v_1 \quad v_2 \quad v_3)^T. \qquad (4.3.5)$$

Hodnota jeho jednotlivých složek vektoru $\vec{v}$ je:

$$v_1 = m_{11}g\,,\; v_2 = -m_3 L_1 L_2 \sin(\Psi)\left(\dot{\Psi}^2 + 2\dot{\Phi}\dot{\Psi}\right),\; v_3 = m_3 L_1 L_2 \sin(\Psi)\dot{\Phi}^2\,. \qquad (4.3.6)$$

Vektor $\vec{T}$ je vektorem řízení a definujeme ho následovně:

$$\vec{T} = \left(F_x \quad M_\Phi \quad M_\Psi\right)^T. \qquad (4.3.7)$$

Nyní máme definovánu dynamiku ramene robotu, pro sestavení stavového popisu ještě musíme definovat stavové proměnné. Jejich počet je šest (tři pro polohu a tři pro rychlost):

$$x_1 = x\,,\; x_2 = \dot{x}\,,\; x_3 = \Phi\,,\; x_4 = \dot{\Phi}\,,\; x_5 = \Psi\,,\; x_6 = \dot{\Psi}\,. \qquad (4.3.8)$$

Nechť F_x je hnací síla v ose *x*, M_Φ je hnací moment v ose Φ a M_Ψ je hnací moment v ose Ψ. Zavedeme tři zobecněné proměnné pro řízení ramene robotu:

$$u_1 = F_x/\left(k_{m1}i_{m1}\right)\,,\; u_2 = M_\Phi/\left(k_{m2}i_{m2}\right),\; u_3 = M_\Psi/\left(k_{m3}i_{m3}\right), \qquad (4.3.9)$$

kde i_m značí zjednodušený převodový poměr mezi zobecněnou silou (resp. momentem) a pohonem, k_m představuje ztrátový poměr. V této úloze volíme i_m = 100, k_m = 0.1 u všech pohonů.

Nyní jsou již zobecněny všechny potřebné proměnné v dynamickém popisu robotu a je možné sestavit jeho stavový popis.

$$\begin{aligned}
&f_1 : \dot{x}_1 = x_2\\
&f_2 : \dot{x}_2 = \left(F_x - m_{11}g\right)/m_{11}\\
&f_3 : \dot{x}_3 = x_4\\
&f_4 : \dot{x}_4 = \left(M_\Phi + m_3 L_1 L_2 \sin x_5\left(x_6^2 + 2x_4 x_6\right) - m_{23}\dot{x}_6\right)/m_{22}\\
&f_5 : \dot{x}_5 = x_6\\
&f_6 : \dot{x}_6 = \left(M_\Psi - m_3 L_1 L_2 x_4^2 \sin x_5 - m_{23}\dot{x}_4\right)/A.
\end{aligned} \qquad (4.3.10)$$

Vidíme, že ve stavovém popisu SCARA robotu se vyskytuje algebraická smyčka. Aby se průběh výpočtu algoritmu zbytečně nezpomalil řešením algebraické smyčky iteračním způsobem, rozhodli jsme se pro řešení analytické. Do výrazu f_4 byl dosazen výraz f_6 a analogicky do výrazu f_6 byl dosazen výraz f_4. Po úpravě jsme dospěli k novému stavovému popisu, který již algebraickou smyčku neobsahuje. Pro zjednodušení výrazu je vhodné zavést dvě pomocné funkce:

$$P\left(x_1,\cdots,x_6,u_1,\cdots,u_3\right),\ Q\left(x_1,\cdots,x_6,u_1,\cdots,u_3\right)\ , \tag{4.3.11}$$

které jsou definované následovně:

$$P = M_\Phi + m_3 L_1 L_2 \sin x_5 \left(x_6^2 + 2x_4 x_6\right),\ Q = M_\Psi - m_3 L_1 L_2 x_4^2 \sin x_5\ . \tag{4.3.12}$$

Stavový popis s vyřešenou algebraickou smyčkou pak vypadá následovně:

$$\begin{aligned}
&f_1 : \dot{x}_1 = x_2\\
&f_2 : \dot{x}_2 = \left(F_x - m_{11} g\right)/m_{11}\\
&f_3 : \dot{x}_3 = x_4\\
&f_4 : \dot{x}_4 = \left(m_{33}P - m_{23}Q\right)/\left(m_{33}m_{22} - m_{23}^2\right)\\
&f_5 : \dot{x}_5 = x_6\\
&f_6 : \dot{x}_6 = \left(m_{22}Q - m_{23}P\right)/\left(m_{33}m_{22} - m_{23}^2\right).
\end{aligned} \tag{4.3.13}$$

Dále zavedeme funkci kritéria optimalizace, jako integrál součtu kvadrátů hodnot řízení:

$$J = \int_{t_0}^{t_1} E\left(x(t),u(t),t\right)dt\ ,\ \text{kde}\ E(x,u,t) = u_1^2 + u_2^2 + u_3^2\ . \tag{4.3.14}$$

Robot obsahuje dva rotační a jeden posuvný člen. Každý rotační člen vnáší do systému velké množství nelinearit. Naštěstí je však v tomto případě posuvný pohyb ramene robotu kolmý ke zbývajícím dvou rotačním pohybům, proto je dynamicky neovlivní. A to nám řešení systému zjednoduší.

Hodnotu stavového vektoru v čase *t* zavedeme stejně jako ve výrazu (4.1.9) a formulujeme okrajové podmínky pro řešení tohoto systému ve tvaru: počáteční čas $t_0 = 0s$, koncový čas $t_1 = 2s$, hodnota stavového vektoru v počátečním čase a koncovém čase:

$$x\left(t_0\right) = \left[0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\right],\ x\left(t_1\right) = \left[1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\right], \tag{4.3.15}$$

což odpovídá v čase t_0 výsunu ramene $x = 0m$, otočení členu 2 ramene $\Phi = 0rad$, úhlu natočení členu 3 ramene $\Psi = 0rad$ a rychlosti všech členů rovnou nule. V čase t_1 pak výsunu ramene $x = 1m$, otočení členu 2 ramene Φ = 1rad ($\Phi \approx 57.3°$), úhlu otočení členu 3 ramene $\Psi = 1rad$ ($\Psi \approx 57.3°$) a rychlosti všech členů rovnou nule.

Tohoto robotu jsme nejprve řešili pomoci gradientní metody 1. řádu. Jako odhad počátečního řízení byl zvolen součet kvadratické, lineární a konstantní funkce ve tvaru:

$$u_n = \alpha_2 \left(t - \alpha_1\right)^2 + \alpha_3 t + \alpha_4 ,\ n = 1\ldots 3 . \qquad (4.3.16)$$

Počáteční odhad konstanty *α* jsme nastavili na hodnotu *α = 1e-9* a při výpočtu byl využit její automatický odhad. Hodnota konstanty penalizace splnění okrajových podmínek byla nastavena na hodnotu *K = 1e6*. Jako počet bodů simulace jsme zvolili hodnotu 101, která se v tomto případě ukázala jako dostačující. Algoritmus dospěl k řešení po 13 350 iteracích a 234 sekundách výpočetního času. Tento typ robotu je již složitější na řešení, ale stále ne tak složitý jako sférický, nebo angulární typ robotu. Proto algoritmus konvergoval k řešení velmi rychle. Získané řešení mělo hodnotu kritéria *J = 17.53566* a normu odchylky od zadaných okrajových podmínek (4.3.15) o velikosti 66e-6.

Dále byla tato úlohu řešena pomocí metody nelineární střelby. Počet bodů simulace byl opět nastaven na hodnotu 101. Tento typ robotu je již složitější a vhodný počáteční odhad pro řešení v zadaném čase je obtížné najít. Pravděpodobně při řešení algebraické smyčky vznikly nelinearity ve stavovém popisu systému, které výpočet touto metodou komplikují. Aby jsme nalezli vhodný počáteční odhad, nastavili jsme nejprve koncový čas simulace na hodnotu t_1 = *0.1s a* odhad vektoru *λ* jsme volili jako *λ = (0, 0, 0, 0, 0, 0)*. Pro tento krátký čas nebyl problém řešení nalézt. Získanou hodnotu vektoru *λ* jsme využili pro další běh metody, při kterém jsme zvýšili velikost koncového času t_1. Takovýmto způsobem jsme se dostali až na řešení pro koncový čas t_1 = *2s*. Pro řešení pak již stačilo pouze 18 sekund výpočetního času a 100 iterací. Kritérium takto získaného řešení je *J = 17.53498* a norma odchylky od okrajových podmínek vyšla 1e-6.

Výsledky řešení SCARA robotu lze nalézt v příloze 4A, kde jsou obsaženy průběhy poloh (na levé straně) a průběhy rychlostí (na pravé straně). V příloze 4B jsou obsaženy průběhy zrychlení (na levé straně) a průběhy řízení (na pravé straně). Modrou barvou jsou znázorněny výsledky získané pomocí gradientní metody 1.řádu, červenou barvou pak výsledky získané pomocí metody nelineární střelby.

Z výsledků je vidět, že řešení získané pomocí obou metod se mírně liší. Přesto je hodnota získaného kritéria přibližně shodná. To je způsobeno tím, že tento typ robotu má na okolí požadovaného řešení velké množství lokálních minim a použité numerické metody mají

sklon konvergovat pouze k lokálnímu minimu. Dosažené lokální minimum pak záleží na zvolení počátečního odhadu. Je vidět, že i přes mírnou odlišnost řízení u_2 získaného pomocí obou metod jsou průběhy polohy daného členu přibližně stejné. Hodnota získaného kritéria se též výrazně neliší.

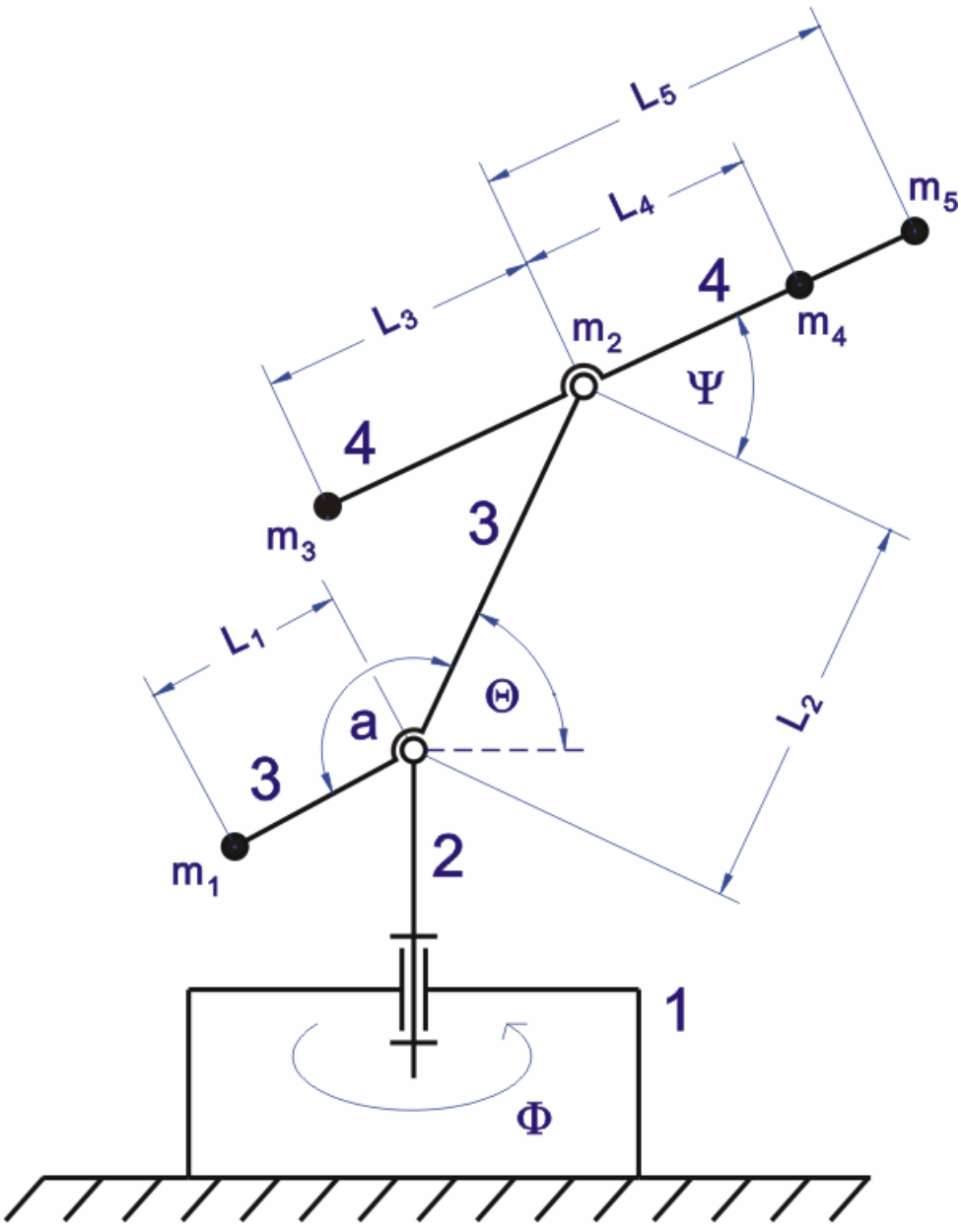
Výsledky řešení obou metod ještě shrneme do následující tabulky. V prvním sloupečku je obsažen název metody, ve druhém doba výpočtu v sekundách. Třetí sloupec obsahuje hodnotu kritéria *J* získanou pomocí dané metody, čtvrtý sloupec obsahuje počet iterací *n* nutných k dosažení řešení. Pátý sloupec obsahuje normu odchylky řešení od okrajových podmínek (4.3.15) a poslední sloupec obsahuje rozlišení dané metody, tedy počet bodů, ve kterých je systém simulován.

metoda	t[s]	J	n	norma	rozlišení
gradientní 1. řádu	234	17.53566	13350	0.000066	101
nelineární střelby	18	17.53497	100	0.000002	101

Tabulka 4.5: Výsledky řešení SCARA robotu.

4.4. Robot pracující v angulárním souřadném systému

Na obrázku 4.5 je uvedeno kinematické schéma robotu s angulárním pracovním prostorem. Orientační ústrojí chapadla je zanedbáno. Uvažujeme pouze polohovací systém se třemi stupni volnosti, tedy rotaci celého těla robotu kolem svislé osy a dva rotační pohyby členů 3 vůči členu 2 a členu 4 vůči členu 3.



Obr. 4.5: Kinematické schéma robotu pracujícího v angulárním souřadném systému.

Nejprve vysvětleme pohyby, které je robot schopen vykonávat a tyto pohyby popišme pomocí proměnných. Člen robotu 2 se vzhledem ke členu 1 pohybuje rotačně kolem svislé osy, úhel natočení popisujeme pomocí proměnné Φ. Člen 3 se pohybuje rotačně vzhledem ke členu 2, popisujeme pomocí proměnné Θ (kývavý pohyb). Člen 4 se vzhledem ke členu 3 pohybuje také rotačně, popisujeme pomocí proměnné Ψ (kývavý pohyb).

Popišme význam jednotlivých parametrů tohoto robotu. Hmotný bod m_4, který je umístěn ve vzdálenosti L_4 od vodorovné osy rotace členu 4 je zobecněnou hmotností zápěstí ramene robotu. Hmotný bod m_5 který je zobecněnou hmotností uchopeného břemene, je umístěn ve vzdálenosti L_5 od vodorovné osy rotace členu 4. Dále pak hmotné body m_1, m_2, m_3 jsou redukované hmotnosti části 3 a 4 ramene robotu. Spolu s jejich vzdálenostmi L_1, vzdálenost od vodorovné osy rotace členu 3, L_2, vzdálenost mezi osami rotace členu 3 a členu 4 (kde m_2 je umístěno v ose rotace členu 4), a L_3, vzdálenost od vodorovné osy rotace členu 4 nahrazují silové působení jednotlivých částí ramene robotu.

Číselné údaje pro konkrétní úlohu jsou:

$$\begin{gathered} m_1 = 34kg,\ m_2 = 16kg,\ m_3 = 39kg,\ m_4 = 35kg,\ m_5 = 20kg, \\ L_1 = 0.9m,\ L_2 = 0.9m,\ L_3 = 0.5m,\ L_4 = 0.6m,\ L_5 = 0.6m, \\ a = 3.1416rad,\ g = 9.81ms^{-2}. \end{gathered} \qquad (4.4.1)$$

Pro zkrácení a zjednodušení tvaru pohybových rovnic je výhodné zavést následující parametry:

$$C_1 = m_1 L_1^2,\ C_2 = L_2 \sum_{i=2}^{5} m_i,\ C_3 = L_2 \sum_{i=3}^{5} m_i L_i,\ C_4 = \sum_{i=3}^{5} m_i L_i^2,\ C_5 = C_1 + C_2. \qquad (4.4.2)$$

Pohybové rovnice robotu pak mají tvar:

$$\begin{aligned} &\ddot{\Phi}\left[C_1 \cos^2(\Theta + a) + C_2 \cos^2\Theta + 2C_3 \cos\Theta \sin(\Theta + \Psi) + C_4 \sin^2(\Theta + \Psi)\right] = \\ &= \dot{\Phi}\dot{\Theta}\left[C_1 \sin(2\Theta + 2a) + C_2 \sin 2\Theta - 2C_3 \cos(\Psi + 2\Theta) - C_4 \sin(2\Psi + 2\Theta)\right] + \\ &+ M_\Phi - \dot{\Psi}\dot{\Phi}\left[2C_3 \cos\Theta \cos(\Psi + \Theta) + C_4 \sin(2\Psi + 2\Theta)\right], \end{aligned} \qquad (4.4.3)$$

$$\ddot{\Theta}\left(2C_3\sin\Psi + C_4 + C_5\right) = M_\Theta - \dot{\Psi}\left(\dot{\Psi} + 2\dot{\Theta}\right)C_3\cos\Psi -$$
$$-0.5\dot{\Phi}^2\left[C_1\sin\left(2\Theta + 2a\right) + C_2\sin 2\Theta - 2C_3\cos\left(\Psi + 2\Theta\right) - C_4\sin\left(2\Psi + 2\Theta\right)\right] -$$
$$-g\left[\left(C_1 / L_1\right)\cos\left(\Theta + a\right) + \left(C_2 / L_2\right)\cos\Theta + \left(C_3 / L_2\right)\sin\left(\Psi + \Theta\right)\right] -$$
$$-\ddot{\Psi}\left(C_3\sin\Psi + C_4\right),$$
$$\ddot{\Psi}C_4 = M_\Psi + \dot{\Theta}^2 C_3\cos\Psi + \dot{\Phi}^2\left[C_3\cos\left(\Psi + \Theta\right) + 0.5C_4\sin\left(2\Psi + 2\Theta\right)\right] -$$
$$-g\left(C_3 / L_2\right)\sin\left(\Psi + \Theta\right) - \ddot{\Theta}\left(C_3\sin\Psi + C_4\right). \qquad (4.4.3)$$

Pro stavový popis ramene robotu definujeme 6 stavových proměnných (tři pro pozici a tři pro rychlost):

$$x_1 = \Phi,\ x_2 = \dot{\Phi},\ x_3 = \Theta,\ x_4 = \dot{\Theta},\ x_5 = \Psi,\ x_6 = \dot{\Psi}. \qquad (4.4.4)$$

Nechť M_Φ je hnací moment v ose Φ, M_Θ je hnací moment v ose Θ, M_Ψ je hnací moment v ose Ψ. Zavedeme tři zobecněné proměnné pro řízení ramene robotu:

$$u_1 = M_\Phi / \left(k_{m1}i_{m1}\right),\ u_2 = M_\Theta / \left(k_{m2}i_{m2}\right),\ u_3 = M_\Psi / \left(k_{m3}i_{m3}\right), \qquad (4.4.5)$$

kde i_m značí zjednodušený převodový poměr mezi zobecněnou silou (resp. momentem) a pohonem, k_m představuje ztrátový poměr. V této úloze volíme i_m = 150, k_m = 0.68 u všech pohonů.

Nyní jsme již zobecnili všechny potřebné proměnné v dynamickém popisu robotu (4.4.3) a můžeme sestavit jeho stavový popis:

$$f_1 : \dot{x}_1 = x_2$$
$$f_2 : \dot{x}_2 = \left\{x_2x_4\left[C_1\sin\left(2x_3 + 2a\right) + C_2\sin 2x_3 - 2C_3\cos\left(x_5 + 2x_3\right) - C_4\sin\left(2x_5 + 2x_3\right)\right] +\right.$$
$$+M_\Phi - x_6x_2\left[2C_3\cos x_3\cos\left(x_5 + x_3\right) + C_4\sin\left(2x_5 + 2x_3\right)\right]\Big\} \Big/$$
$$\Big/\left[C_1\cos^2\left(x_3 + a\right) + C_2\cos^2 x_3 + 2C_3\cos x_3\sin\left(x_3 + x_5\right) + C_4\sin^2\left(x_3 + x_5\right)\right]$$
$$f_3 : \dot{x}_3 = x_4 \qquad (4.4.6)$$

$$f_4 : \dot{x}_4 = \{M_\Theta - x_6(x_6 + 2x_4)C_3 \cos x_5 -$$
$$-0.5x_2^2\left[C_1 \sin(2x_3 + 2a) + C_2 \sin 2x_3 - 2C_3 \cos(x_5 + 2x_3) - C_4 \sin(2x_5 + 2x_3)\right] -$$
$$-g\left[(C_1 / L_1)\cos(x_3 + a) + (C_2 / L_2)\cos x_3 + (C_3 / L_2)\sin(x_5 + x_3)\right] -$$
$$-\dot{x}_6(C_3 \sin x_5 + C_4)\}/(2C_3 \sin x_5 + C_4 + C_5)$$

$$f_5 : \dot{x}_5 = x_6$$

$$f_6 : \dot{x}_6 = \{M_\Psi + x_4^2 C_3 \cos x_5 + x_2^2\left[C_3 \cos(x_5 + x_3) + 0.5C_4 \sin(2x_5 + 2x_3)\right] -$$
$$-g(C_3 / L_2)\sin(x_5 + x_3) - \dot{x}_4(C_3 \sin x_5 + C_4)\}/C_4 \,. \qquad (4.4.6)$$

Ve stavovém popisu lze opět nalézt výskyt algebraické smyčky. Úlohu s algebraickou smyčkou není možné řešit přímo. Proto je nutné tuto smyčku stejně jako u cylindrického a SCARA robotu odstranit. Aby se průběh výpočtu algoritmu zbytečně nezpomalil řešením algebraické smyčky iteračním způsobem, rozhodli jsme se opět pro řešení analytické. Do výrazu f_4 byl dosazen výraz f_6 a analogicky do výrazu f_6 byl dosazen výraz f_4. Po úpravě jsme dospěli k novému stavovému popisu, který již algebraickou smyčku neobsahuje. Pro zjednodušení výrazu byly zavedeny čtyři pomocné funkce:

$$P(x_1,\cdots,x_6,u_1,\cdots,u_3),\ Q(x_1,\cdots,x_6,u_1,\cdots,u_3),\ R(x_1,\cdots,x_6),\ W(x_1,\cdots,x_6), \qquad (4.4.7)$$

které jsou definované následovně:

$$Q = \{M_\Theta - x_6(x_6 + 2x_4)C_3 \cos x_5 -$$
$$-0.5x_2^2\left[C_1 \sin(2x_3 + 2a) + C_2 \sin 2x_3 - 2C_3 \cos(x_5 + 2x_3) - C_4 \sin(2x_5 + 2x_3)\right] -$$
$$-g\left[(C_1 / L_1)\cos(x_3 + a) + (C_2 / L_2)\cos x_3 + (C_3 / L_2)\sin(x_5 + x_3)\right]\},$$

$$P = \{M_\Psi + x_4^2 C_3 \cos x_5 + x_2^2\left[C_3 \cos(x_5 + x_3) + 0.5C_4 \sin(2x_5 + 2x_3)\right] -$$
$$-g(C_3 / L_2)\sin(x_5 + x_3)\}$$

$$W = C_3 \sin(x_5 + C_4),$$

$$R = 2C_3 \sin x_5 + C_4 + C_5 \,. \qquad (4.4.8)$$

Stavový popis systému robotu pracujícího v angulárním souřadném systému s vyřešenou algebraickou smyčkou pak vypadá následovně:

$$f_1 : \dot{x}_1 = x_2$$

$$f_2 : \dot{x}_2 = \left\{ x_2 x_4 \left[C_1 \sin\left(2x_3 + 2a\right) + C_2 \sin 2x_3 - 2C_3 \cos\left(x_5 + 2x_3\right) - C_4 \sin\left(2x_5 + 2x_3\right) \right] + \right.$$
$$\left. + M_\Phi - x_6 x_2 \left[2C_3 \cos x_3 \cos\left(x_5 + x_3\right) + C_4 \sin\left(2x_5 + 2x_3\right) \right] \right\} \Big/$$
$$\Big/ \left[C_1 \cos^2\left(x_3 + a\right) + C_2 \cos^2 x_3 + 2C_3 \cos x_3 \sin\left(x_3 + x_5\right) + C_4 \sin^2\left(x_3 + x_5\right) \right]$$

$$f_3 : \dot{x}_3 = x_4$$

$$f_4 : \dot{x}_4 = \left(\mathrm{C_4Q - P \cdot W}\right) \Big/ \left(\mathrm{C_4R - W^2}\right)$$

$$f_5 : \dot{x}_5 = x_6$$

$$f_6 : \dot{x}_6 = \left(\mathrm{P \cdot R - Q \cdot W}\right) \Big/ \left(\mathrm{C_4R - W^2}\right). \qquad (4.4.9)$$

Dále zavedeme funkci kritéria optimalizace, jako integrál součtu kvadrátů hodnot řízení

$$J = \int_{t_0}^{t_1} E\left(x(t), u(t), t\right) dt \text{ , kde } E(x,u,t) = u_1^2 + u_2^2 + u_3^2 . \qquad (4.4.10)$$

Robot pracující v angulárním pracovním prostoru je silně nelineární. Má tři rotační členy a žádný z pohybů jeho členů neleží v rovině kolmé k pohybu ostatních členů. Proto jsou všechny jeho stavové rovnice vzájemně provázány a vzhledem k existenci pouze rotačních členů, také velmi rozsáhlé a složité. To se projevuje velkými problémy při řešení tohoto robotu. Je zapotřebí velkého množství iterací, dostatečně velkého rozlišení simulace a k tomu odpovídajícímu velkému množství výpočetního času. Právě u tohoto typu robotu jsou kladeny vysoké nároky na dobrý počáteční odhad. Při volbě špatného počátečního odhadu není ani jedna z metod schopna najít řešení a výpočet selhává.

Hodnotu stavového vektoru v čase *t* zavedeme stejně jako ve výrazu (4.1.9) a formulujeme okrajové podmínky pro řešení tohoto systému ve tvaru: počáteční čas t_0 = *0s*, koncový čas t_1 = *1s*, hodnota stavového vektoru v počátečním čase a koncovém čase:

$$x\left(t_0\right) = \left[0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0\right],\ x\left(t_1\right) = \left[1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0\right], \qquad (4.4.11)$$

což odpovídá v čase t_0 výsunu otočení ramene okolo svislé osy Φ = *0rad*, kyvu členu 3 ramene Θ = *0rad*, kyvu členu 4 ramene Ψ = *0rad* a rychlosti všech členů rovnou nule. V

čase t_1 pak otočení ramene okolo svislé osy $\Phi = 1rad$ ($\Phi \approx 57.3°$), kyvu členu 3 ramene $\Theta = 1rad$ ($\Theta \approx 57.3°$), kyvu členu 4 ramene $\Psi = 1rad$ ($\Psi \approx 57.3°$).

Tohoto robotu jsme nejprve řešili pomoci gradientní metody 1. řádu. Jako odhad počátečního řízení již nešlo použít nulové konstantní funkce. Proto jsme použili konstantní funkce o hodnotách, které udrží rameno robotu v počáteční (nulové) poloze jednotlivých členů po celou dobu řízení, tedy od počátečního do koncového času. Hodnoty těchto funkcí jsou definovány ve tvaru: $u_n(t) = k_n$, kde $n = 1, 2, 3$, kde k_n je vhodně zvolená konstanta. Počáteční odhad konstanty α byl nastaven na hodnotu $\alpha = 1e\text{-}9$ a při výpočtu byl využit její automatický odhad. Hodnota konstanty penalizace splnění okrajových podmínek byla nastavena na hodnotu $K = 1e10$. Tato kombinace hodnot se jevila jako nejvhodnější. Výpočet je při ní dostatečně rychlý a metoda konverguje k řešení pouze s malou normou odchylky od splnění okrajových podmínek. Počet bodů simulace jsme byli nuceni nastavit na poměrně vysokou hodnotu 202, protože při využití nižšího rozlišení simulace výpočet selhával.

Algoritmus dospěl k řešení po 15 000 iteracích a 450s výpočetního času. Získané řešení mělo kriterium $J = 45.0896$ a normu odchylky od splnění okrajových podmínek o velikosti 0.00003. Rychlost výpočtu je i přes velkou složitost tohoto typu robotu vysoká. To je způsobeno tím, že byl zvolen, na rozdíl od ostatních řešených úloh, kratší čas řešení o velikosti jedné sekundy. Tento kratší časový interval byl zvolen zejména proto, aby jsme získané řešení mohli porovnat s řešením získaným pomocí metody nelineární střelby, protože u této metody je velmi obtížné najít vhodný počáteční odhad pro delší časy řešení.

Dále byla tato úloha řešena pomocí metody nelineární střelby. Počet bodů simulace byl opět nastaven na hodnotu 202 a počet iterací na hodnotu 300. Jako počáteční odhad vektoru λ v tomto případě nebylo možné zvolit nulový, nebo jednotkový vektor, výpočet pak selhával. Ani pouhé experimentování s dosazováním náhodných čísel do počátečního odhadu vektoru λ nepřineslo žádné výsledky. Proto jsme byli nuceni použít následující metody odhadu. Nejprve byl nastaven koncový čas řešení na hodnotu $t_1 = 0.1s$, pro takto krátký čas není problém najít řešení ani při velmi špatném počátečním odhadu. Výsledný vektor λ jsme použili jako počáteční odhad pro řešení s koncovým časem $t_1 = t_1 + \Delta t$, kde Δt je nějaký malý časový interval, např. $\Delta t = 0.05s$. Takto lze pokračovat až k řešení

požadované úlohy. Po nalezení vhodného počátečního odhadu lze snadno řešit úlohu i pro jiné hodnoty požadovaného stavového vektoru $x(t_1)$ v koncovém čase t_1.

S vhodným počátečním odhadem algoritmus našel řešení po 300 iteracích a 193s výpočetního času. Kritérium takto získaného řešení je *J = 45.059904* a norma odchylky od okrajových podmínek vyšla 1e-6. Je vidět, že tato metoda při nalezení vhodných počátečních podmínek konverguje k řešení velmi rychle a s konstantní délkou kroku.

Výsledky řešení robotu pracujícího v angulárním souřadném systému lze nalézt v příloze 5A, kde jsou obsaženy průběhy poloh (na levé straně) a průběhy rychlostí (na pravé straně). V příloze 5B jsou obsaženy průběhy zrychlení (na levé straně) a průběhy řízení (na pravé straně). Modrou barvou jsou znázorněny výsledky získané pomocí gradientní metody 1.řádu, červenou barvou pak výsledky získané pomocí nelineární střelby. Díky tomu, že pro tuto úlohu byl zvolen krátký simulační čas nevznikala ani u takto složitého typu robotu velká numerická chyba a jednotlivé průběhy se přibližně překrývají.

Výsledky řešení obou metod shrneme do následující tabulky. V prvním sloupečku je název metody, ve druhém doba výpočtu v sekundách. Třetí sloupec obsahuje hodnotu kritéria *J* získanou pomocí dané metody, čtvrtý sloupec obsahuje počet iterací *n* nutných k dosažení řešení. Pátý sloupec obsahuje normu odchylky řešení od splnění okrajových podmínek (4.4.11) a poslední sloupec obsahuje rozlišení dané metody, tedy počet bodů, ve kterých je systém simulován.

metoda	t[s]	J	n	norma	rozlišení
gradientní 1. řádu	450	45.089600	15000	0.000030	202
nelineární střelby	193	45.059904	300	0.000001	202

Tabulka 4.6: Výsledky řešení robotu pracujícího v angulárním souřadném systému.

Rovněž bylo provedeno porovnání gradientní metody 1. řádu realizované v jazyce C++ s výsledky získanými gradientní metodou 1. řádu realizovanou v prostředí MATLAB. Porovnání je uvedeno v následující tabulce.

metoda	t[s]	J	n	norma	rozlišení
gradientní C++	450	45.089600	15000	0.000030	202
gradientní MATLAB	2930	45.620104	16500	0.000145	169

Tabulka 4.7: Porovnání gradientní metody 1. řádu realizované v C++ a MATLAB.

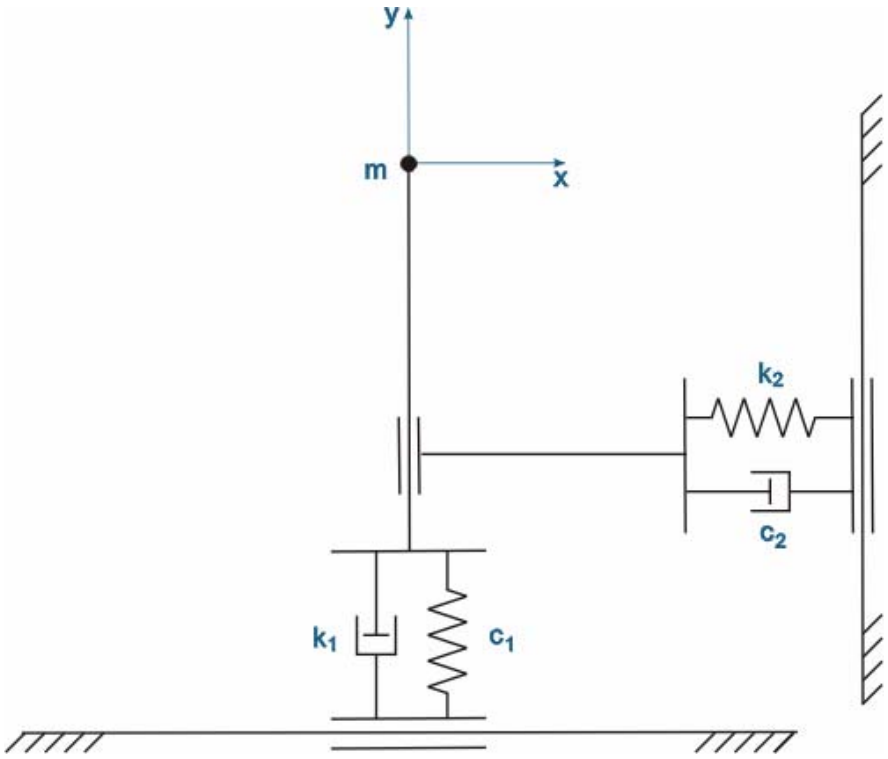
Porovnání výsledků obou metod získaných při řešení tohoto robotu lze nalézt v příloze 5C, kde jsou obsaženy průběhy poloh (na levé straně) a průběhy rychlostí (na pravé straně). V příloze 5D jsou obsaženy průběhy zrychlení (na levé straně) a průběhy řízení (na pravé straně). Modrou barvou jsou znázorněny výsledky získané pomocí gradientní metody 1. řádu realizované v jazyce C++, červenou barvou pak výsledky získané pomocí gradientní metody 1. řádu realizované v prostředí MATLAB.

Z porovnání si můžeme všimnout, že metoda realizovaná v jazyce C++ je zhruba 7x rychlejší (a to i přes vyšší rozlišení simulace) než metoda realizovaná v MATLAB. Také si můžeme všimnout, že je zde poměrně velký rozdíl v získaných kriteriích řešení a i z grafického porovnání vidíme, že průběhy se nepřekrývají. To je způsobeno tím, že při vyšší rychlosti výpočtu metody realizované v jazyce C++ jsme si mohli dovolit vyšší rozlišení simulace a dosáhnout tak přesnějšího výpočtu. Nevýhodou numerických metod je, že v průběhu výpočtu dochází ke vzniku numerických chyb, které se v průběhu jednotlivých iterací kumulují. Metoda pak může konvergovat k lokálnímu minimu nějaké jiné funkce, která vznikne z funkce původní působením chyb. Toto se také stalo při řešení pomocí metody realizované v prostředí MATLAB. Proto se zde vyskytla poměrně velká odchylka obou řešení.

5. Optimální řízení v prostoru se statickými překážkami

Metody kterými jsme se do této doby zabývali řešily dané systémy bez ohledu na omezující podmínky jednotlivých stavových veličin. V praxi je však s tímto omezením nutno počítat. V pohybu robotu se například mohou vyskytnout překážky, nebo je rozsah pohybu jeho členů omezen na nějaký interval. Právě proto jsme se v rámci této práce zabývali způsobem modifikace gradientní metody 1. řádu pro řešení úlohy optimálního řízení v prostoru se statickými překážkami. Po dohodě s vedoucím diplomové práce jsme se rozhodli řešit pouze jednu jednoduchou nelineární úlohu se dvěma stupni volnosti a se statickou překážkou typu kruh.

5.1. Formulace jednoduché úlohy



Obr. 5.1: Kinematické schéma jednoduché úlohy se dvěma stupni volnosti

Jako systém který vyhovuje výše zmíněným požadavkům byl zvolen hmotný bod se dvěma stupni volnosti a to posuvný pohyb ve směru osy *x* a posuvný pohyb ve směru osy *y*. Ve směrech jednotlivých os na hmotný bod působí vždy síla pružiny, síla nelineárního tlumiče rychlosti a dále pak jednotlivé síly řízení u_1 (působí ve směru osy *x*), u_2 (působí ve směru osy *y*). Počátek os *x* a *y* byl zvolen v rovnovážné poloze, kdy na hmotný bod působí nulová síla z obou pružin. Parametry úlohy: koeficient tuhosti pružin $k_1 = 5Nm^{-1}$, $k_2 = 11.5Nm^{-1}$, koeficient tlumení nelineárních tlumičů rychlosti $c_1 = 0.1Nm^{-2}s^2$, $c_2 = 0.25Nm^{-2}s^2$, hmotnost hmotného bodu $m = 1kg$. Kinematické schéma této úlohy je zobrazeno na obrázku 5.1.

Dynamický popisu systému byl sestaven takto:

$$m\ddot{x} = -c_1\dot{x}^2 - k_1x + u_1$$

$$m\ddot{y} = -c_2\dot{y} - k_2y + u_2. \tag{5.1}$$

Systém popíšeme pomocí čtyř stavových proměnných: poloha hmotného bodu x_1 ve směru osy *x,* jeho rychlost x_2 ve směru osy *x* a poloha hmotného bodu x_3 ve směru osy *y* a jeho rychlost x_4 ve směru osy *y*. Soustava stavových rovnic pak bude mít následující tvar:

$$\begin{aligned}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= \left(-c_1x_2^2 - k_1x_1 + u_1\right)\big/m \\
\dot{x}_3 &= x_4 \\
\dot{x}_4 &= \left(-c_2x_4^2 - k_2x_3 + u_2\right)\big/m.
\end{aligned} \tag{5.2}$$

Funkce kritéria pro optimalizaci byla definována následovně:

$$J = \int_{t_0}^{t_1}\left(u_1^2 + u_2^2\right)dt. \tag{5.3}$$

Zbývá formulovat okrajové podmínky řešení tohoto systému: počáteční čas $t_0 = 0s$, koncový čas $t_1 = 1s$, stav systému v počátečním čase $x_1(t_0) = 0m$, $x_2(t_0) = 0ms^{-1}$, $x_3(t_0) = 0m$, $x_4(t_0) = 0ms^{-1}$ a stav systému v koncovém čase $x_1(t_1) = 1m$, $x_2(t_1) = 0ms^{-1}$, $x_3(t_1) = 1m$, $x_4(t_1) = 0ms^{-1}$.

Jako statická překážka byl zvolen kruh, jehož hraniční křivkou je kružnice popsaná pomocí vztahu (5.4). Proměnné *a, b* jsou souřadnice středu kružnice a proměnná *r* je její poloměr. Pro překážku byly zvoleny následující parametry: *a = 0.5m, b = 0.5m* a poloměr *r = 0.35m*.

$$\Phi : (x-a)^2 + (y-b)^2 - r^2 = 0 \tag{5.4}$$

5.2. Teoretické základy řešení úloh s omezením

Po prostudování [1] jsme zjistili, že s výhodou můžeme použít teorii o řešení jednostranných variací. Jedná se úlohu nalezení extrému funkcionálu, kde od třídy přípustných křivek požadujeme splnění další podmínky a to, že žádná z těchto křivek nesmí procházet žádným vnitřním bodem oblasti *R* ohraničené pomocí uzavřené křivky $\Phi(x,y) = 0$, ale může obsahovat části křivek, které jsou hranicí oblasti *R*. Můžou tedy nastat dvě možnosti a to, že extrémála vůbec neprochází oblastí *R,* pak můžeme použít metody výpočtu beze změny. V druhém případě však vzniká nová situace. Na částech hranice oblasti *R* jsou možné jen jednostranné variace, protože dovnitř oblasti nemohou přípustné křivky zasahovat. Pro zbývající části křivky, které probíhají vně oblasti *R* opět platí první možnost a to proto, že variujeme danou křivku pouze v oblasti, kde jsou možné oboustranné variace, které jsme schopni řešit.

Řešení této úlohy tedy může nastat pouze na křivkách skládajících se z částí hranice oblasti *R* a z částí extrémál mimo tuto oblast. Pro sestrojení křivky realizující extrém musíme najít hladké napojení těchto částí. Jedná se tedy o řešení úlohy s volným koncovým, resp. počátečním bodem, který leží na hranici oblasti *R*. Zjistili jsme dva způsoby, kterými můžeme stávající gradientní metodu upravit pro řešení této úlohy. Jedním způsobem je samotnou metodu vůbec neměnit, ale pouze realizovat její nadstavbu. Tato nadstavba nejprve zjistí průsečík extrémál s oblastí *R* a pak prohledává jeho okolí, přičemž zjišťuje hodnoty kritéria a vyhledá bod nejlepšího napojení, tak aby toto napojení bylo hladké. Druhou možností je úprava samotné gradientní metody, tedy zadání souřadnic koncového, resp. počátečního bodu pomocí analytické funkce křivky ohraničující oblast *R* do funkce penalizace okrajových podmínek $G(x(t_1))$ definované ve vztahu (2.3). V této práci se budeme zabývat pouze řešením úlohy se statickou překážkou za pomoci prohledávání okolí průsečíku.

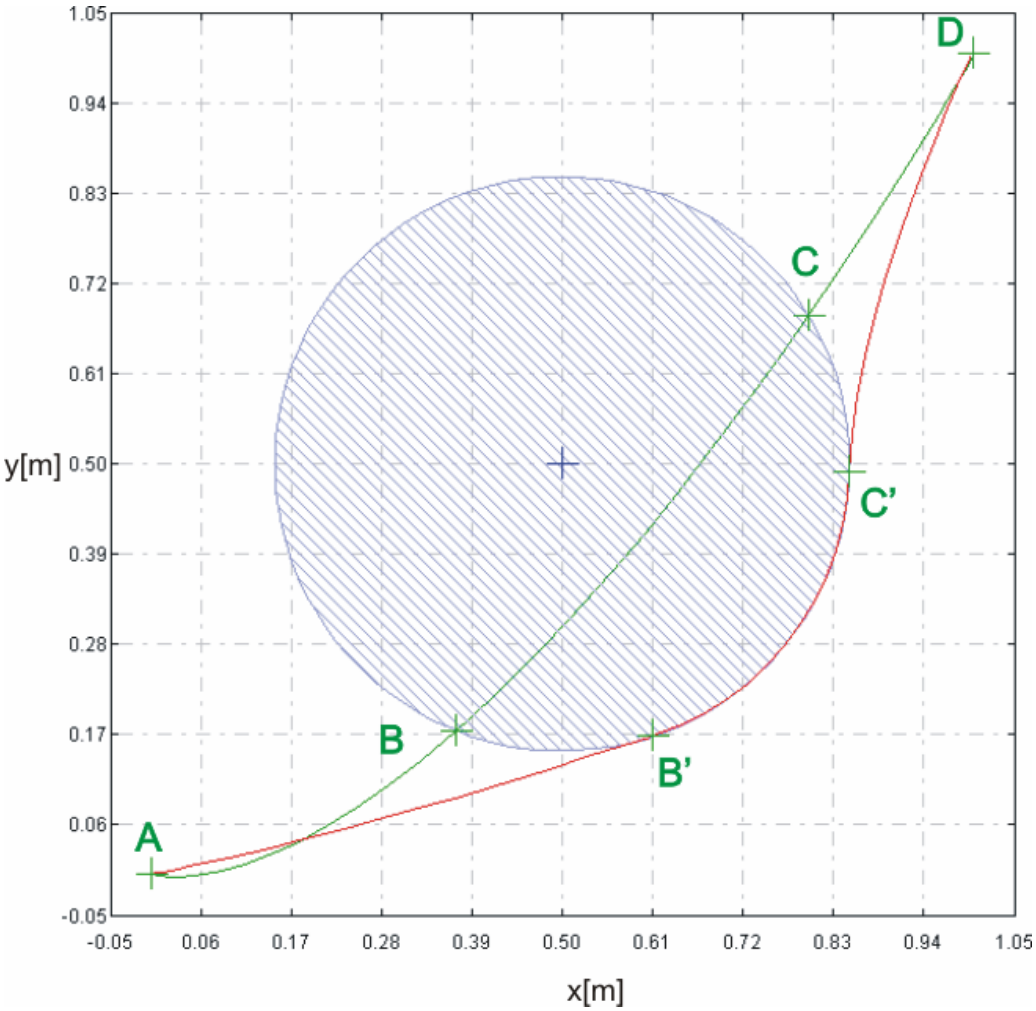
5.3. Realizace úpravy gradientní metody

Principem této metody je vytvoření nadstavby gradientní metody. Princip je následující. Nejprve je zjištěno optimální řešení bez ohledu na překážky. Dále jsou zjištěny body ve kterých došlo k protnutí hraniční křivky oblasti *R*. Pokud žádný průsečík neexistuje, pak algoritmus končí. V opačném případě je trajektorie rozdělena na tři části. První část probíhá od počátečního bodu k nejbližšímu průsečíku, druhá část probíhá mezi body průsečíku po hranici oblasti *R* a třetí část probíhá od druhého průsečíku do koncového bodu. Úkolem algoritmu je najít takové řízení, aby hodnota kritéria první a třetí části trajektorie byla minimální možná, přičemž koncový, resp. počáteční bod části 1, resp. části 3 trajektorie je na hranici oblasti *R* volný. Jeho poloha se na hranici oblasti *R* může libovolně měnit tak, aby výsledné kritérium bylo co nejnižší, ale musí být splněna podmínka, že směr rychlosti bodu systému, na jehož polohu klademe omezení (např. efektoru ramene robotu), je shodný se směrem tečny hranice oblasti *R* v daném bodě. Body průsečíků jsou brány jako počáteční odhad. Algoritmus ukončí svojí činnost, pokud je vybrán takový koncový, resp. počáteční bod, který má nejmenší možnou hodnotu kritéria *J*. Při volbě směru prohledávání okolí průsečíků je také zohledněn fakt, aby druhá část trajektorie ležící na hraniční křivce oblasti *R* byla nejkratší možná. Algoritmus vyhledává takovým způsobem, že v každém kroku, kdy je zjištěn pokles hodnoty kritéria *J* řešení se bod na kružnici posune o úhel *da* a pokud je tendence kritéria *J* klesající, hodnota *da* se zdvojnásobí (pro zrychlení vyhledávání). Po zjištění přesáhnutí minima se algoritmus vrátí zpět a změní se způsob prohledávání. Nyní se již hodnota *da* nezvětšuje, ale zmenšuje se až do doby, než je dosaženo řešení s dostatečnou přesností (*da* je dostatečně malé).

Výsledek řešení jednoduché úlohy touto metodou vidíte na následujícím obrázku 5.2. Modře se šrafy je znázorněna oblast překážky *R*. Zelenou barvou křivky je zobrazena optimální trajektorie hmotného bodu v souřadnicích *x, y* z bodu A do bodu D bez ohledu na překážku. Body B a C vyjadřují průsečíky s překážkou.

Algoritmus v jednotlivých krocích prohledává okolí bodů průsečíků B a C. Pro každý z nich vypočítává optimální řízení a odpovídající hodnotu funkce kritéria *J*. Takovýmto způsobem byly zjištěny body B’ a C’, ve kterých je splněna vlastnost hladkého napojení na kružnici a zároveň je toto místo napojení optimální. Křivka označená červenou barvou znázorňuje

optimální průběh polohy hmotného bodu v souřadnicích *x, y* s ohledem na statickou překážku. Řešení bylo dosaženo za 24 sekund výpočetního času. Hodnota kritéria první části trajektorie má hodnotu $J_1 = 20.22$ a hodnota třetí části trajektorie je $J_3 = 34.53$.



Obr. 5.2: Optimální průběhy trajektorií hmotného bodu s ohledem na statické překážky

Tento způsob výpočtu se projevil jako poměrně rychlý. Při prohledávání okolí průsečíků je sice nutné vypočítat velké množství optimálních řízení do, resp. z jednotlivých bodů okolí průsečíků, ale v tomto případě můžeme vždy, již získané řešení využít jako počáteční odhad pro výpočet příštího bodu. Vzhledem k faktu, že následují bod při vyhledávání není příliš vzdálen od původního bodu, je výpočet velmi rychlý. Tuto metodu by také bylo možné velice jednoduše rozšířit na řešení úlohy v trojrozměrném prostoru, kdy místo kruhu by překážka byla reprezentována jako válec.

6. Vizualizace

Grafy průběhů poloh, rychlostí, zrychlení a řízení jednotlivých členů ramene robotu v čase, které reprezentují dosažené výsledky optimálního řízení, nám sice umožňují ověřit funkci a vytvořit vzájemné porovnání jednotlivých optimalizačních metod, ale již nám příliš neumožňují vytvořit představu o charakteru pohybu ramene robotu a o tvaru trajektorie koncového bodu (efektoru). Právě z tohoto důvodu jsme se rozhodli vytvořit prostředí pro simulaci jednotlivých základních struktur robotů (cylindrický, sférický, angulární a SCARA robot), které by bylo schopno tyto roboty vhodně zobrazit. Jednak jako řezy ve 2D a dále jako virtuální model ve 3D, spolu s trajektorií efektoru ramene robotu.

Pro realizaci tohoto prostředí jsme opět (stejně jako pro realizaci optimalizačních metod) zvolili Visual C++ .NET. Jako vizualizační systém jsme použili OpenGL (tvůrce SGI) a to zejména z důvodu, že je tento systém poměrně snadno implementovatelný, platformě nezávislý a navíc je již velice dobře podporován ovladači většiny výrobců grafických akcelerátorů.

6.1. Základní vlastnosti OpenGL

Velkou výhodou tohoto systému na rozdíl od Direct-X je garance funkčnosti jakéhokoliv základního rysu uvedeného ve specifikaci jádra. To je zajištěno tím, že veškeré základní funkce jsou zabudované v softwarovém renderu a pokud aplikace volá funkci, která není implementována akceleračním hardwarem, pak je automaticky volána její softwarová verze. Do ovladačů je také umožněno implementovat softwarově operace, které by měl provádět hardware. Ovšem s podmínkou, že daná softwarově realizovaná operace bude rychlejší než stejná operace provedená pomocí hardware.

OpenGL je velice silný nástroj, který nám umožňuje následují funkce:

- vykreslení pouze viditelných hran (tzv. z-buffer)
- veškeré transformace objektů ve scéně
- nastavení parametrů kamery snímající scénu
- definování světelných zdrojů mnoha typů (ambientní, bodové, směrové, kužel) ve scéně a nastavení mnoha jejich vlastností (ambientní složka, difúzní složka, …)

- vytváření dynamických seznamů objektů pro zrychlení vykreslení scény
- aplikování textur na objekty
- speciální efekty (mlha, odrazy, difúze, …)
- antialiasing

OpenGL je schopné vykreslit pouze základní jednoduchá primitiva jako jsou: body, úsečky, trojúhelníky a polygony. S tím je třeba při návrhu simulačního prostředí počítat a vytvořit vhodné datové struktury pro uchování objektů jednotlivých částí ramene robotu.

Vzhledem k tomu, že je OpenGL systémově nezávislá knihovna, nepodporuje práci s okny. Umožňuje nám pouze vykreslení dané plochy, ale zachytávání a obsluhu zpráv musíme zařídit sami. Proto bylo při realizaci simulačního prostředí využito knihoven MFC v kombinaci se systémem OpenGL.

6.2. Princip použití OpenGL

Celá scéna se pomocí OpenGL vykresluje do paměťového bufferu a následně je jako jeden obrázek překreslena na obrazovku. Tímto způsobem se zabraňuje blikání, ke kterému by jinak docházelo v důsledku postupného vykreslování jednotlivých objektů. Vykreslování spočívá v následujících operacích. Nejprve vytvoříme vhodnou transformaci (posunutí, rotace, změna měřítka), dále do systému OpenGL předáme definice materiálů vykreslovaného objektu spolu se základními primitivy (body, úsečky a polygony), které daný objekt charakterizují. Takto definovaný objekt je pak přímo vyrenderován do paměťového bufferu (a to s ohledem na z-buffer, tedy hloubku kterou mají ve scéně).

Jak jsme se již výše zmínili OpenGL umí vykreslovat pouze základní primitiva jako jsou bod, úsečka, trojúhelník a polygon. Nyní si popíšeme, jakým způsobem se tyto primitiva do systému zadávají. Nejprve použijeme příkaz *glBegin (mode)*, kde *mode* je typ primitiva a může nabývat následující hodnoty *GL_POINTS* (definuje objekt zobrazený pouze z bodů), *GL_LINES* (definuje objekt jako úsečku), *GL_TRIANGLE* (definuje 3D trojúhelníkovou plošku) a další. Dále definujeme body ze kterých se objekt skládá pomocí příkazu *glVertex3d (x, y, z)*, kde proměnné *x, y, z* jsou souřadnice polohy bodu v prostoru. Barvu bodu definujeme pomocí příkazu *glColor3f (r, g, b)*, kde proměnné *r, g, b* jsou hodnoty jednotlivých složek RGB modelu barev a nabývají hodnot v rozsahu 0 až 1.

Pokud definujeme trojúhelník, nebo polygon, můžeme buď použít automaticky vypočtenou normálu celé plochy, nebo zadat pro každý vrchol normálu zvlášť. Tato informace je nutná pro hladké stínování. Ukončení definice primitiva označíme pomocí příkazu *glEnd*.

Pro reprezentaci složitějších objektů (sestavených ze základních primitiv a definic materiálů jejich povrchů) nám OpenGL umožňuje využít funkcí pro tvorbu dynamických seznamů. Pro vytvoření takovéhoto seznamu použijeme příkaz *glNewList (n,GL_COMPILE)*, kde n je číslo daného seznamu. Takovýchto seznamů lze vytvořit velké množství omezené pouze pamětí počítače. Dále pak definujeme jednotlivá primitiva pomocí příkazů *glBegin* až *glEnd.* Konec dynamického seznamu označíme pomocí příkazu *glEndList*. Kdykoli pak chceme daný objekt vykreslit, stačí pouze použít funkci volání daného seznamu *glCallList (n)*, kde *n* je číslo požadovaného seznamu a objekt je vykreslen na místo zadané pomocí zvolené transformace.

Některé základní objekty jsou obsaženy v externí knihovně *glaux.h*. Mezi tyto objekty patří koule, krychle, kvádr, torus, prstenec, kužel, a také objekt „konvice“. Jednotlivé metody této knihovny generují objekty jako posloupnost polygonů (drátěný model), nebo jako plně vykreslené těleso včetně jeho povrchů.

6.3. Vytvoření virtuálních modelů

Sestavovat jednotlivé objekty robotu ze základních primitiv je příliš komplikované a časově velice náročné. Proto bylo vhodnější vytvořit program, který by byl schopen importovat jednotlivé objekty z 3D vizualizačního programu 3D Studia MAX. Tento postup je velice univerzální, protože lze model celého robotu navrhnout v profesionálním programu k tomuto určenému a to s výhodou všech jeho pomocných nástrojů. Navíc v případě nutnosti jakýchkoliv změn danou část robotu jednoduše upravíme v prostředí 3D Studia MAX a importujeme do našeho prostředí, bez jakékoliv změny v programu (pokud nedojde ke změně měřítek objektu). Samozřejmě musíme při návrhu robotu dbát na fakt, aby vytvořený model nebyl příliš složitý (nebyl sestaven z příliš velkého počtu polygonů), protože to by běh simulačního prostředí velice zpomalovalo.

Program realizovaný pro import objektů pracuje s ASCII typem souboru, který je 3D Studio Max schopné exportovat. Tento typ souboru je velice výhodný, protože ho jsme schopni jednoduše analyzovat (veškeré informace v něm obsažené jsou textového

charakteru) a převést do vlastní datové struktury. Takto realizovaný import je schopný převést libovolné množství objektů (sestavených z polygonů) a to dokonce včetně jejich materiálů.

Pro uchování jednotlivých částí ramene robotu byla použita následující datová struktura. Na začátku je umístěna hlavní hlavička, která obsahuje informace o celkovém počtu objektů ve struktuře a o celkovém počtu materiálů. Následuje pole materiálů, které obsahuje informace o jejich vlastnostech: ambientní, difúzní, odražená barva a lesk. Dále pokračuje hlavička objektu číslo jedna, která obsahuje informace o počtu bodů (vertex) a plošek (face), ze kterých se daný objekt skládá a odkaz na číslo materiálu v poli materiálů. Následuje pole bodů a pole plošek objektu číslo jedna. Datová struktura pokračuje hlavičkou objektu dvě až hlavičkou posledního objektu opět spolu s polem jeho bodů a plošek.

HLAVNÍ HLAVIČKA
počet objektů .. *N* počet materiálů .. *m*
pole materiálů o velikosti *m*
HLAVIČKA OBJEKTU *1*
počet bodů .. b_1 pošet plošek .. p_1 číslo materialu .. m_1
pole bodů objektu *1* o velikost b_1
pole plošek objektu *1* o velikost p_1
…
HLAVIČKA OBJEKTU *N*
počet bodů .. b_N pošet plošek .. p_N číslo materialu .. m_N
pole bodů objektu *N* o velikost b_N
pole plošek objektu *N* o velikost p_N

Obr. 6.1: Datová struktura pro uložení definic objektů vizualizace

Tyto struktury jsou uloženy v souboru s příponou *d3d* pro každou část ramene robotu. Z datové struktury je vidět, že se každá část ramene robotu může skládat z libovolného počtu objektu a také libovolného počtu materiálů těchto objektů. Toto schéma je velmi univerzální a umožňuje definovat širokou škálu tvarů.

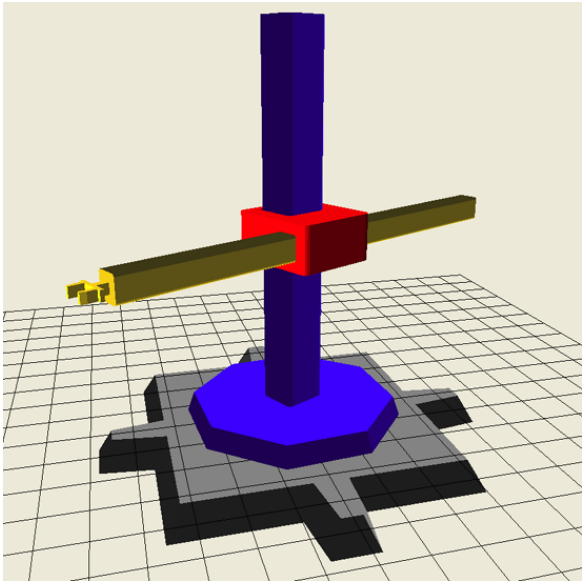
6.4. Vlastnosti simulačního prostředí

Simulační prostředí je navrženo jako jeden program tvořený čtyřmi samostatnými okny. Program po otevření simulačních dat vyhodnotí typ řešené úlohy a podle něj pak automaticky nastaví obsah jednotlivých oken.

Nejprve si popíšeme funkci jednotlivých oken prostředí. První okno obsahuje ovládací prvky pro řízení chodu simulace. Zde můžeme načítat libovolná data, měnit rychlost přehrávání a nastavovat zobrazení jednotlivých komponentů systému. Druhé okno slouží pro zobrazení scény ve 2D a to jednak v řezu, vedeným tělem ramene robotu, a pohledu shora. Dále pro zobrazení aktuálních hodnot poloh, rychlostí a řízení jednotlivých členů ramene robotu. Třetí okno zobrazuje trojrozměrný pohled na virtuální scénu ramene robotu. Veškeré zobrazovací funkce v tomto okně byly realizovány pomocí OpenGL. Scénou můžeme libovolně rotovat, posouvat a měnit její měřítko, dále můžeme zobrazit trajektorii koncového bodu ramene robotu (efektoru). Ve čtvrtém okně jsou zobrazeny okrajové podmínky pro řešení dané úlohy řízení, typ robotu jehož řešení zobrazujeme a hodnota kritéria *J* dosaženého řešení.

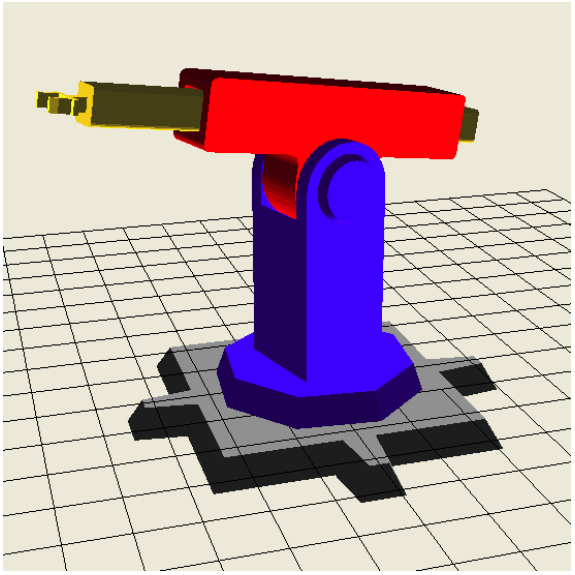
Prostředí podporuje čtyři typy základních struktur robotů. Jedná se o SCARA robot a roboty pracující v cylindrických, sférických a angulárních souřadnicích. Jednotlivé virtuální modely základních struktur robotů jsou vyobrazeny na následujících stranách spolu s jejich popisem.

Na obrázku 6.2 lze nalézt virtuální model robotu pracujícího v cylindrickém souřadném systému. Podle schématu popsaném v kapitole 4.1. je šedou barvou označen člen 1 (základna robotu), modrou barvou člen 2 (rotace okolo svislé osy), červenou barvou je zobrazen člen 3 (posuvný pohyb nahoru a dolů) a nakonec žlutou barvou je označen člen 4 a 5 (výsun ramene).



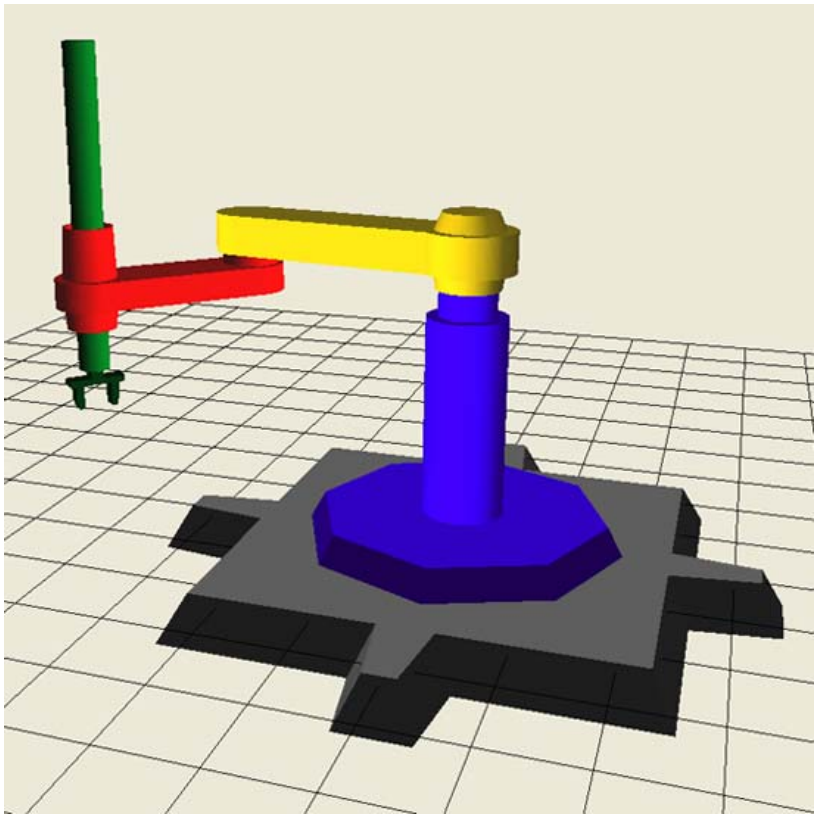
Obr. 6.2: Virtuální model robotu pracujícího v cylindrickém souřadném systému

Na spodním obrázku je zobrazen virtuální model robotu pracujícího ve sférickém souřadném systému. Podle schématu popsaném v kapitole 4.2. je šedou barvou označen člen 1 (základna robotu), modrou barvou člen 2 robotu (rotace okolo svislé osy), červenou barvou je zobrazen člen 3 (kývavý pohyb vůči členu 2) a nakonec žlutou barvou je označen člen 4 (výsun ramene).



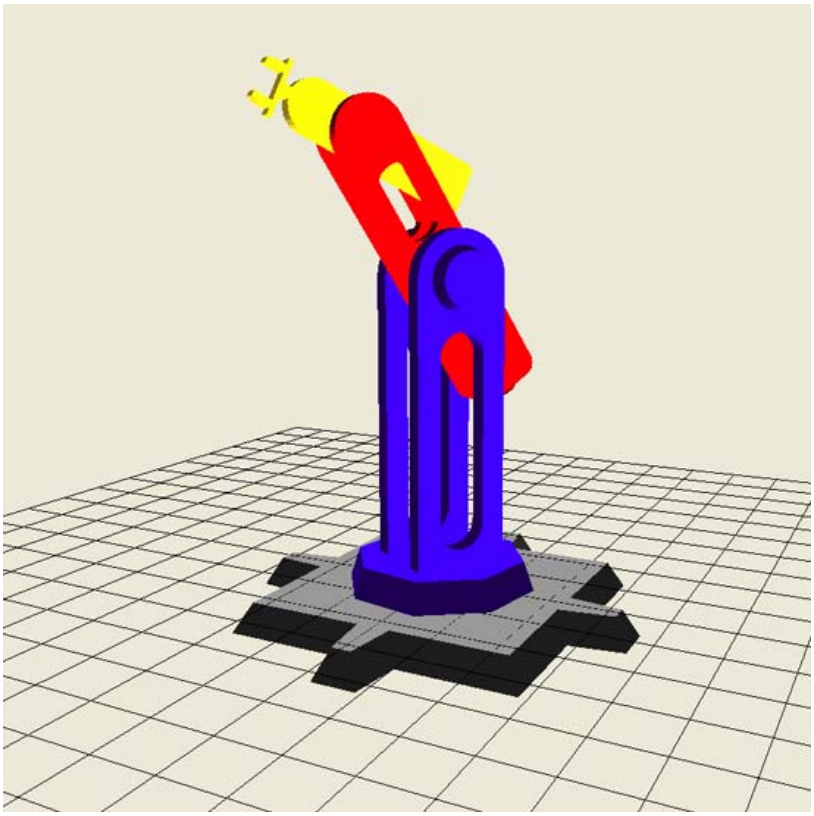
Obr. 6.3: Virtuální model robotu pracujícího ve sférickém souřadném systému

Na následujícím obrázku je zobrazen virtuální model SCARA robotu. Podle schématu popsaném v kapitole 4.3. je šedou barvou označena základna robotu, modrou barvou člen 1 robotu pevně spojený se základnou, žlutou barvou je zobrazen člen 2 (rotace vůči členu 1 kolem svislé osy), červenou barvou je znázorněn člen 3 (rotace vůči členu 2 kolem svislé osy) a zeleně je zobrazen poslední člen 4 ramene robotu (posun nahoru a dolů).



Obr. 6.4: Virtuální model SCARA robotu

Na posledním obrázku 6.5 je zobrazen virtuální model robotu pracujícího v angulárním souřadném systému. Podle schématu popsaném v kapitole 4.4. je šedou barvou označen první člen ramene robotu (základna), modrou barvou člen 2 (rotační pohyb kolem svislé osy), červenou barvou je zobrazen člen 3 (kývavý pohyb vůči členu 2) a žlutou barvou je znázorněn člen 4 (kývavý pohyb vůči členu 3) .



Obr. 6.5: Virtuální model robotu pracujícího v angulárním souřadném systému

6.5. Vizualizace trajektorie

Důležitým úkolem také bylo zobrazení trajektorie koncového bodu ramene robotu (efektoru), které nám přispívá k lepšímu pochopení prováděného pohybu. Po načtení dat ze souboru se trajektorie generuje podle vztahů pro polohu efektoru (v souřadném systému OpenGL) v závislosti na poloze jednotlivých členů ramene robotu.

Pro polohu efektoru cylindrického robotu (kapitola 4.1.) platí následující vztahy:

$$x = -(L_6 + r)\sin(\Phi) - e\cos(\Phi),$$

$$y = (L_6 + r)\cos(\Phi) - e\sin(\Phi), \qquad (6.1)$$

$$z = h + z.$$

Poloha efektoru sférického robotu (kapitola 4.2.) je definována jako:

$$x = \left(e\sin\Psi - \left(L_5 + x\right)\cos\Psi\right)\sin\Phi,$$

$$y = \left(-e\sin\Psi + \left(x + L_5\right)\cos\Psi\right)\cos\Phi, \tag{6.2}$$

$$z = \left(L_5 + x\right)\sin\Psi + e\cos\Psi.$$

Poloha efektoru SCARA robotu (kapitola 4.3.) je definována následovně:

$$x = L_1\cos\Phi + L_2\cos\left(\Psi + \Phi\right),$$

$$y = L_1\sin\Phi + L_2\sin\left(\Psi + \Phi\right), \tag{6.3}$$

$$z = L_3 - x.$$

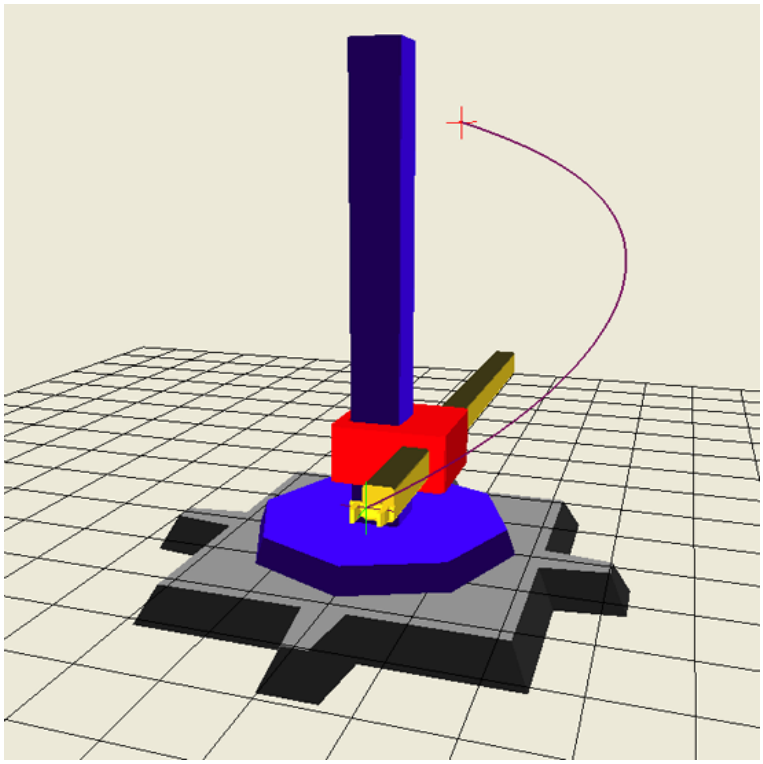
Pro polohu efektoru angulárního robotu (kapitola 4.4.) platí následující vztahy:

$$x = -\left(L_2\cos\Theta + L_5\sin\left(\Theta + \Psi\right)\right)\sin\Phi,$$

$$y = \left(L_2\cos\Theta + L_5\sin\left(\Theta + \Psi\right)\right)\cos\Phi, \tag{6.4}$$

$$z = -L_5\cos\left(\Theta + \Psi\right) + L_2\sin\Theta.$$

Pomocí těchto vztahů je vypočítáno pole bodů a ty pak předány do OpenGL, kde z nich vytvoříme základní objekt typu lomená čára, který je spolu se všemi objekty ve scéně vykreslován. Aby byla trajektorie lépe viditelná, byla pro její vykreslení nastavena tloušťka úsečky (příkaz *glLineWidth)* na hodnotu 2. Poloha efektoru je znázorněna pomocí křížku umístěného mezi kleštinu robotu. V simulačním prostředí můžeme také nastavit vykreslení pouze trajektorie se zobrazeným efektorem, který se po ní pohybuje. Počáteční bod trajektorie je zobrazen pomocí křížku zelené barvy a koncový bod trajektorie je označen pomocí křížku červené barvy. Na obrázku 6.5 je zobrazen virtuální model cylindrického robotu se zapnutou funkcí zobrazení trajektorie efektoru.



Obr. 6.6: Virtuální model robotu pracujícího v cylindrickém souřadném systému včetně trajektorie.

7. Závěr

Cílem diplomové práce bylo navrhnout a vytvořit nástroj pro optimální řízení základních struktur robotů, jako je cylindrický, sférický, angulární a SCARA robot, pomocí metod dynamické optimalizace, založených na variačním počtu. Pro tuto práci byla zvolena gradientní metoda 1. řádu a metoda nelineární střelby. Dále, pro ověření optimálních řešení získaných pomocí obou metod, bylo vytvořeno simulační prostředí založené na systému OpenGL. Toto prostředí umožňuje zobrazení trajektorie efektoru a také průběhu pohybů základních struktur robotů ve 3D, čímž přispívá k lepšímu porozumění chování řízeného systému. Obě metody byly spolu se simulačním prostředím, pro maximální urychlení běhu programu, implementovány v moderním vývojovém prostředí Visual C++ verze 7.0.

Funkčnost metod byla ověřena při řešení jednoduché úlohy a výsledky porovnány s analytickým řešením této úlohy, přičemž zjištěná řešení byla přibližně shodná. Obě metody byly také samozřejmě odzkoušeny na všech výše zmíněných základních strukturách robotů a konvergovaly k přibližně shodným výsledkům. Metoda nelineární střelby se nejlépe projevila při řešení robotů pracujících v cylindrickém a sférickém souřadném systému a to hlavně vzhledem k velmi vysoké rychlosti konvergence k optimálnímu řešení. Na druhou stranu, gradientní metoda 1. řádu vykázala lepší výsledky při řešení angulárního a SCARA robotu, kde sice konvergovala pomaleji, ale nastavení vhodného počáteční odhadu bylo výrazně jednodušší a rychlejší než u metody nelineární střelby. Ukázalo se, že gradientní metoda 1. řádu je schopna stabilně řešit i tak složitý systém jako je angulární robot a to v poměrně širokém rozsahu pohybů jeho jednotlivých členů. Tato metoda byla také úspěšně modifikována pro optimální řízení jednoduché úlohy se dvěma stupni volnosti v prostoru se statickou překážkou typu kruh.

Velkou výhodou těchto metod je jejich univerzálnost. Realizace obou metod byla provedena takovým způsobem, že je lze velice rychle pozměnit pro řešení jiného typu úloh a to nejenom z oblasti robotiky. Tyto metody je totiž možné použít i pro řešení takových typů úloh jako je např. nalezení optimální trajektorie raketoplánu, modulu přistávajícího na povrch Měsíce, družice, nebo letadla.

Literatura

[1] Elsgolc L. E.: Variační počet, Praha, SNTL, 1965.

[2] Bryson A. E., Ho Yu-Chi: Applied Optimal Control, New York, John Wiley & sons, 1975.

[3] Brunovský P.: Matematická teória optimálneho riadenia, Bratislava, Alfa, 1980.

[4] Žára J., Beneš B., Felkel P.: Moderní počítačová grafika, Praha, Computer Press, 1998.

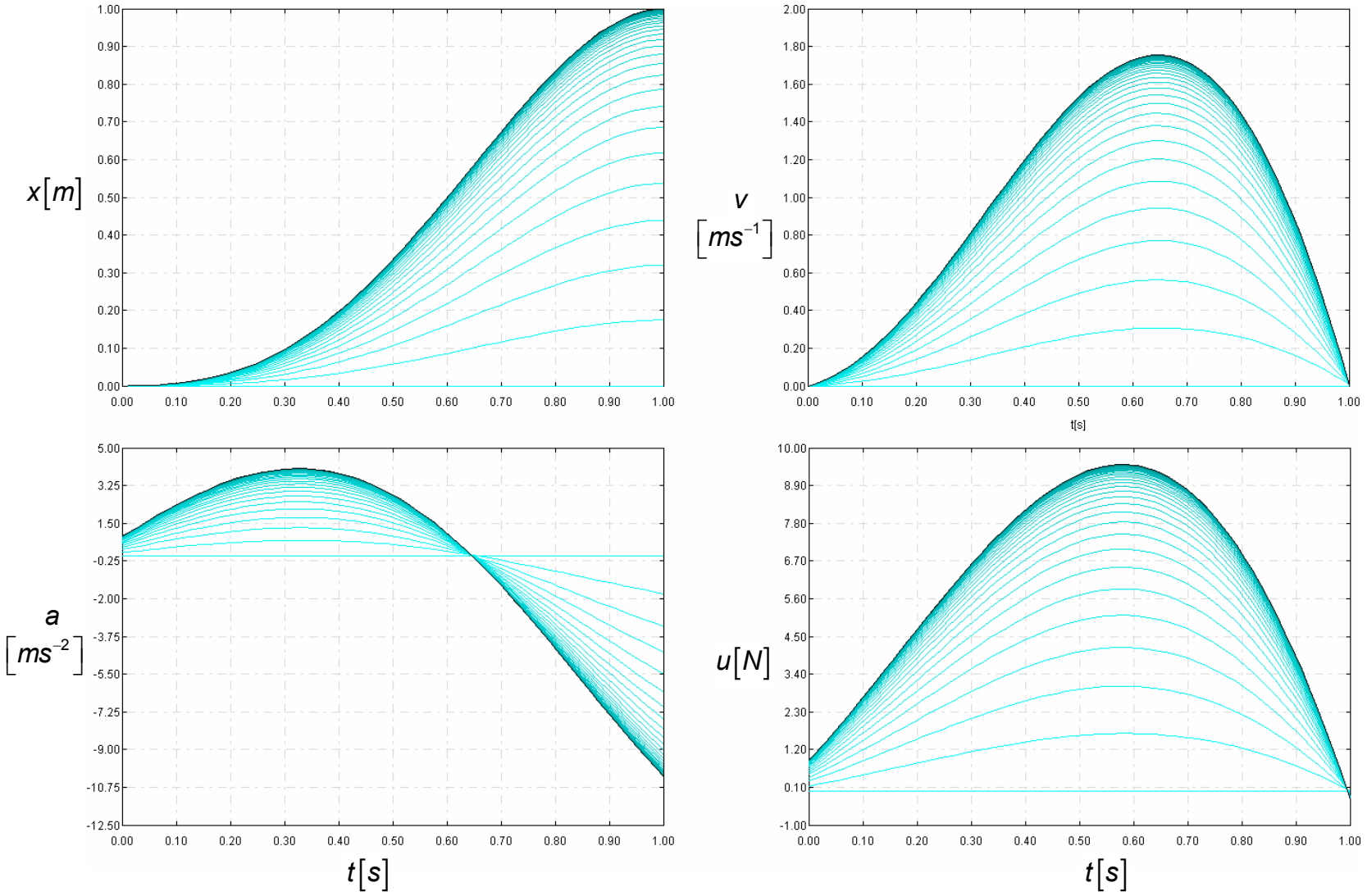
[5] Kruglinski J. D.: Mistrovství ve Visual C++, Praha, Computer Press, 1999.

[6] Lewis F. L., Abdallah C. T., Dawson D. M.: Control of robot manipulators, New York, Macmillan, 1993.

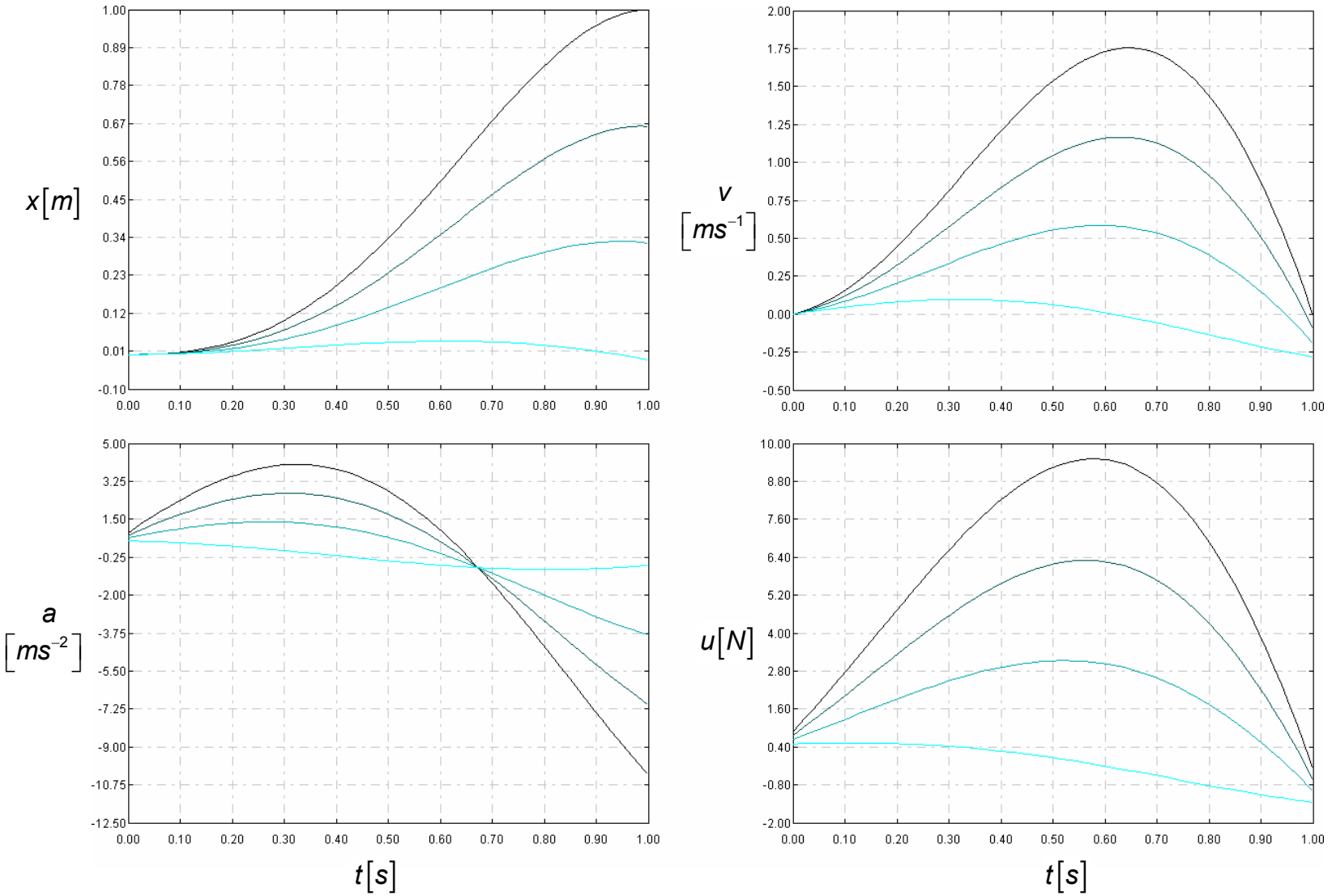
[7] Jesse L.: C++ Unleashed, United States of America, Sams, 1999.

[8] Fosner R.: OpenGL Programming for Windows 95 and Windows NT, San Diego, Addison-Wesley Professional, 1996.

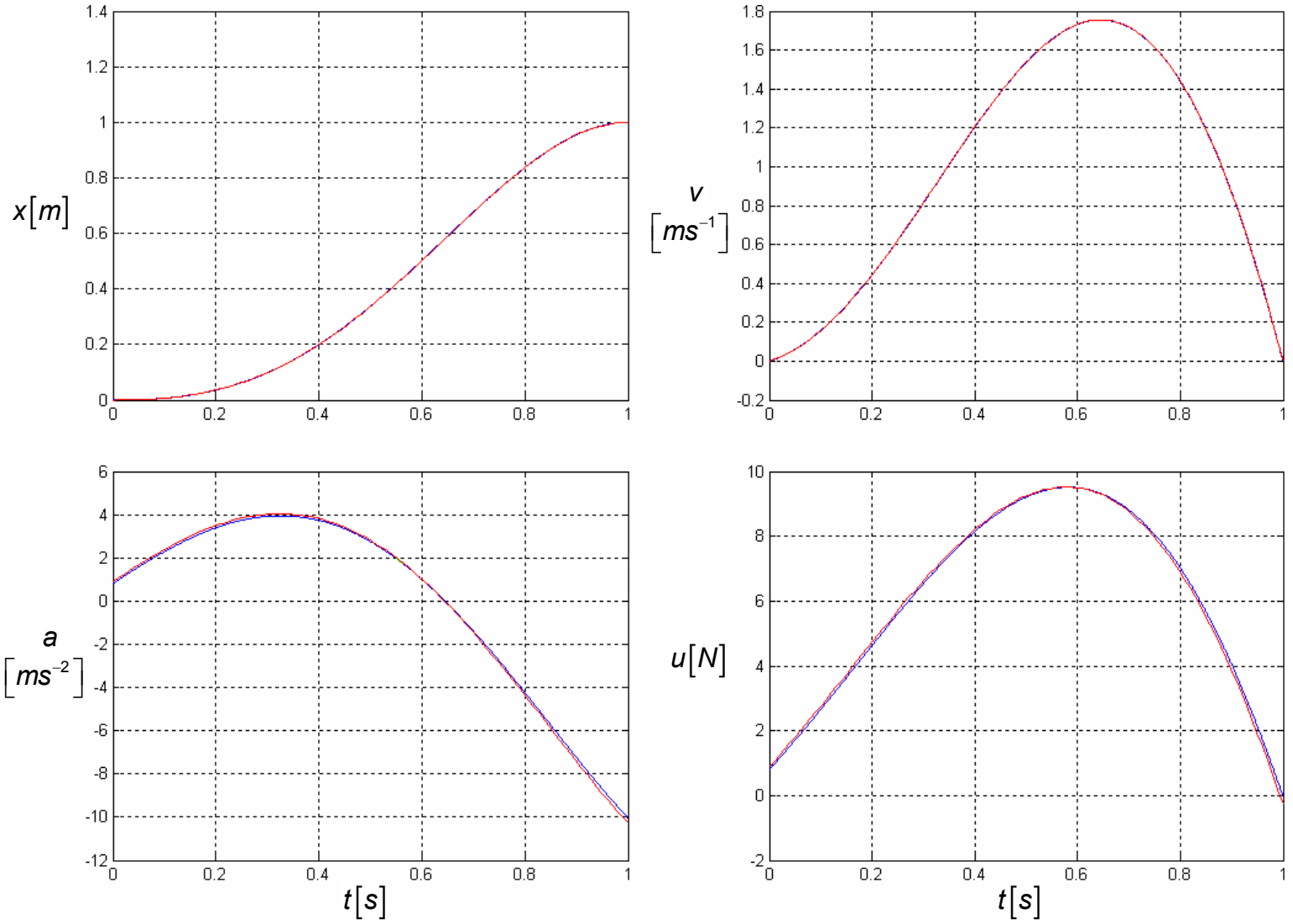
Přílohy



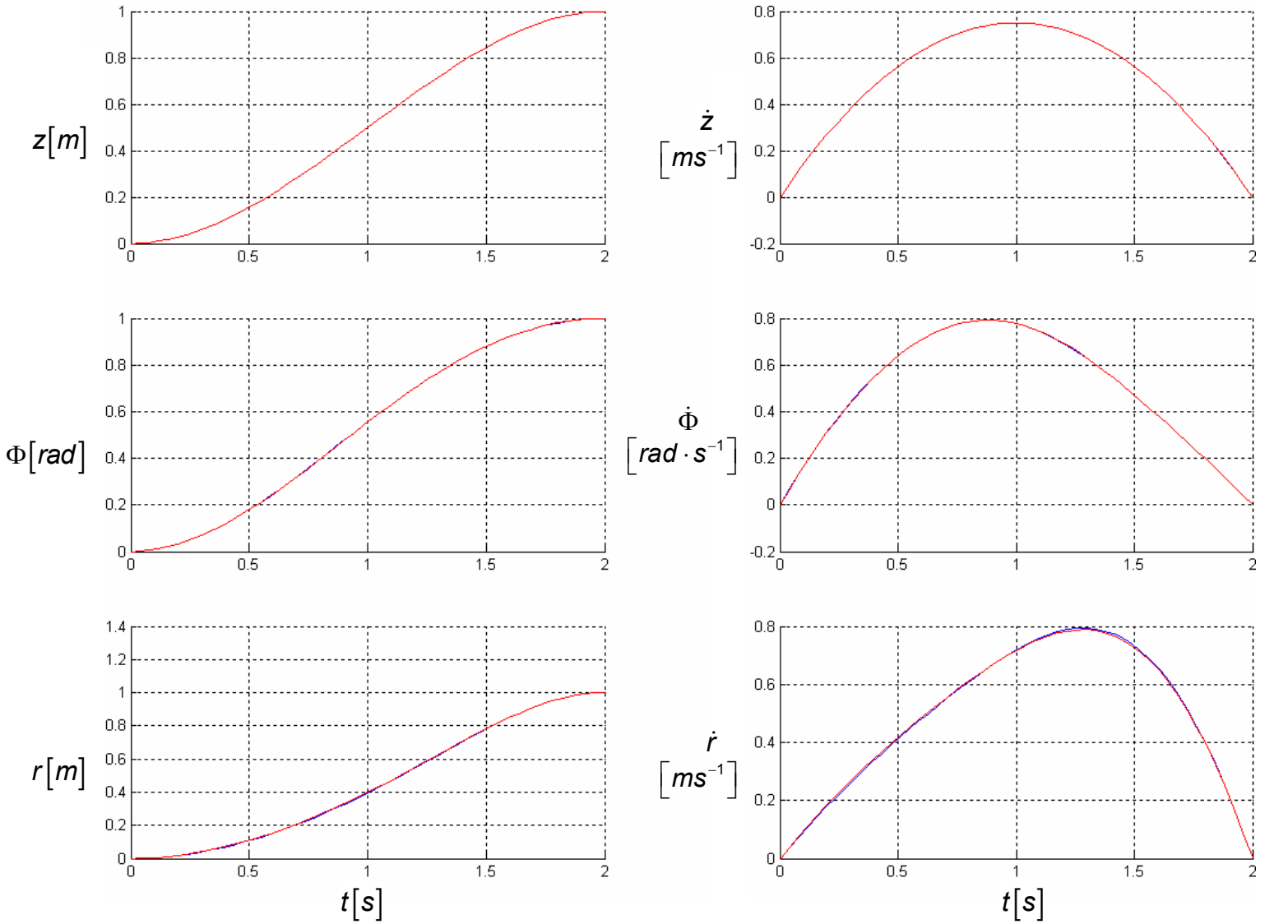
PŘÍLOHA 1A – průběh řešení gradientní metodou 1. řadu v jednotlivých iteracích 0 80



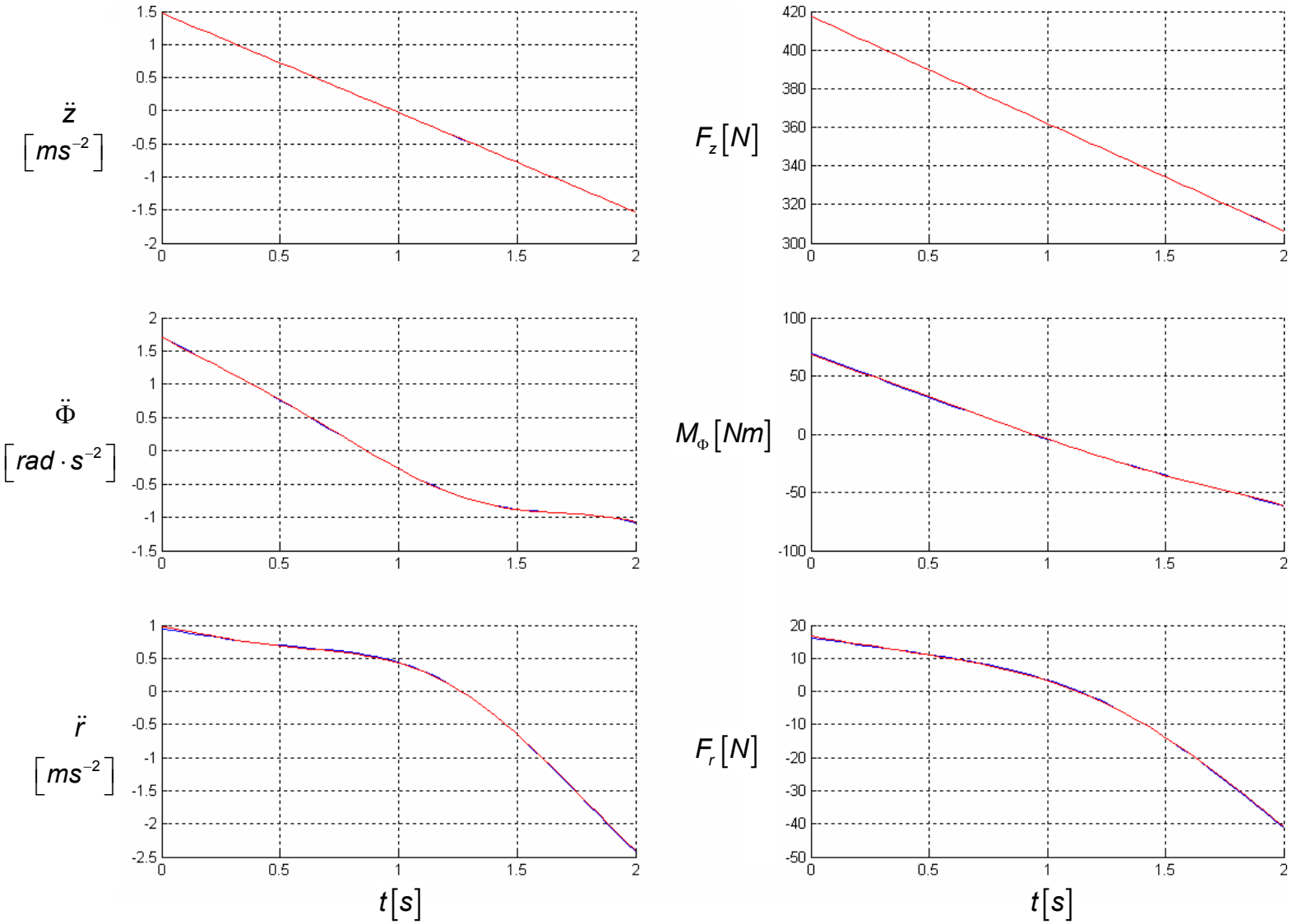
PŘÍLOHA 1B – průběh řešení metodou nelineární střelby v jednotlivých iteracích 0 3



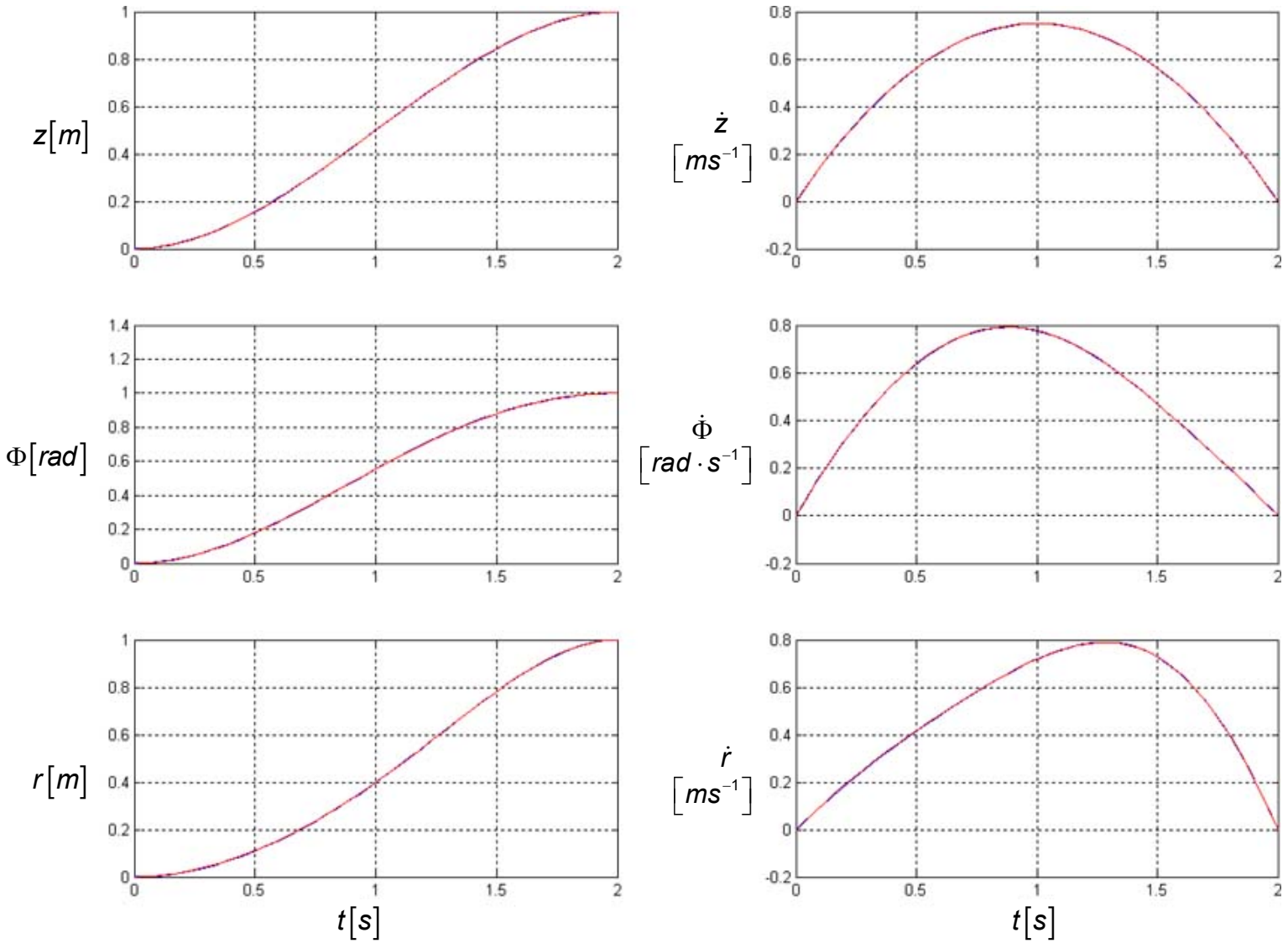
PŘÍLOHA 1C – porovnání řešení získaných pomocí analytické metody, gradientní metody 1.řádu, metody nelineární střelby



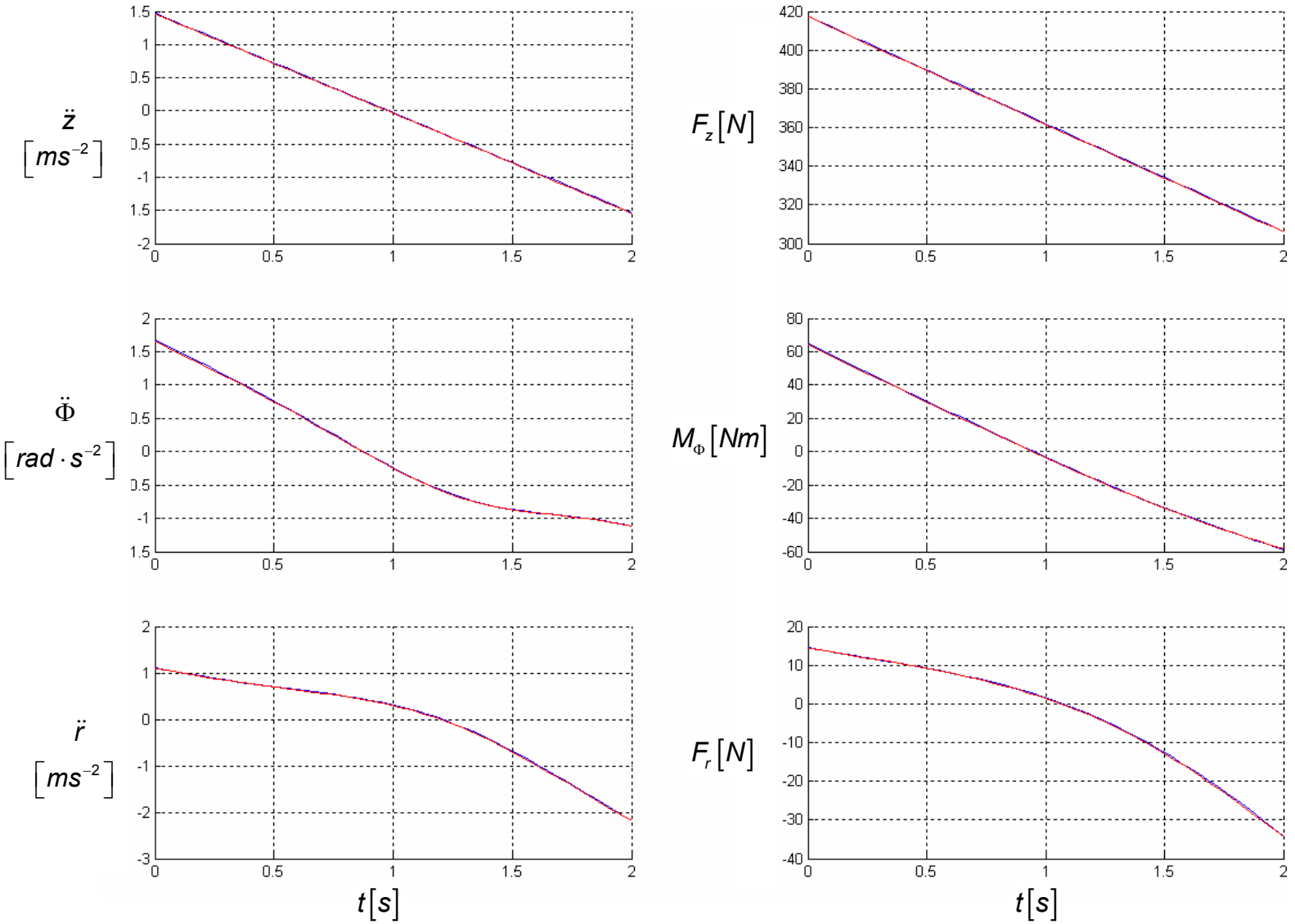
PŘÍLOHA 2A – porovnání řešení cylindrického robotu pomocí gradientní metody 1.řádu a metody nelineární střelby



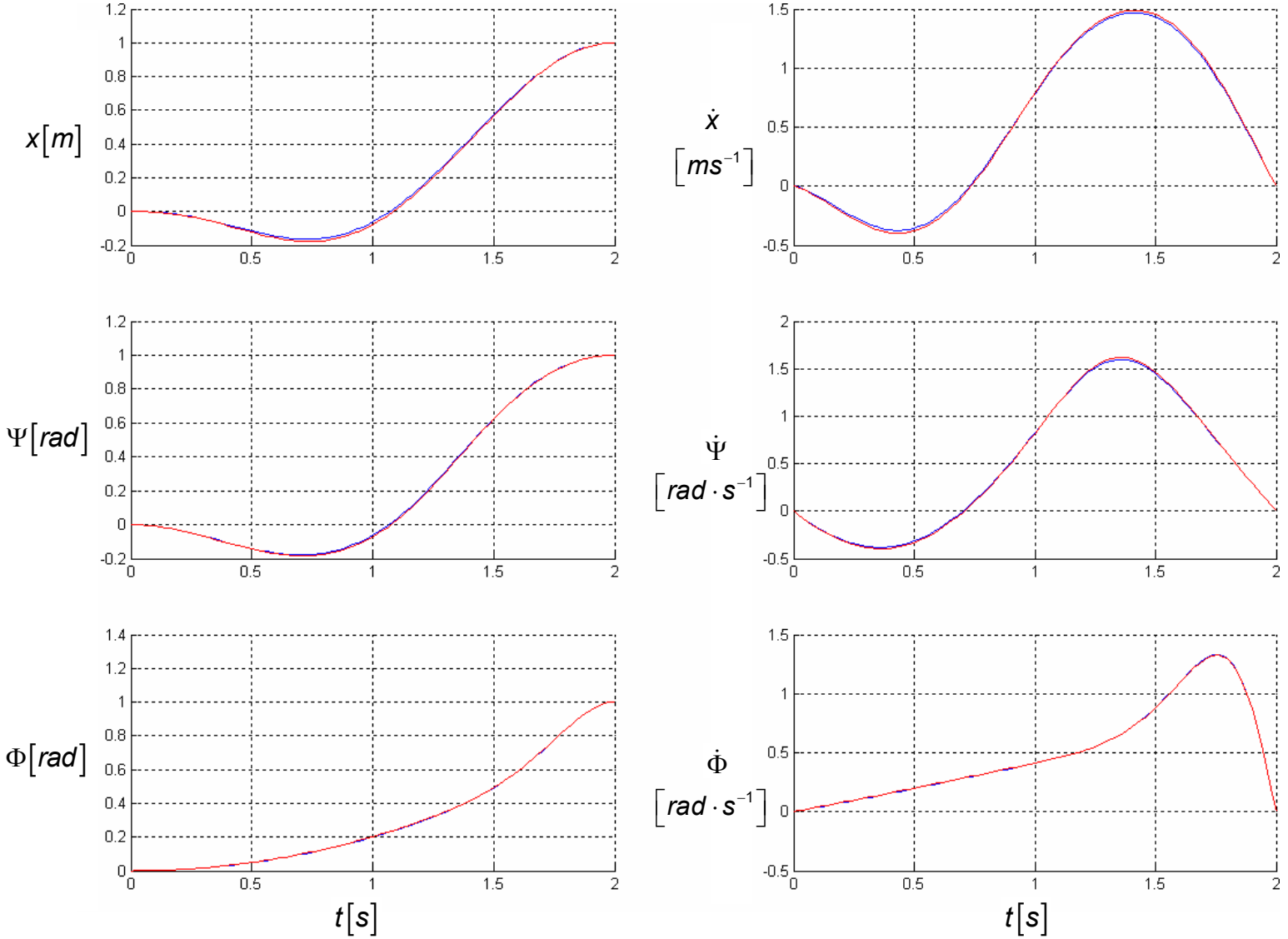
PŘÍLOHA 2B – porovnání řešení cylindrického robotu pomocí gradientní metody 1.řádu a metody nelineární střelby



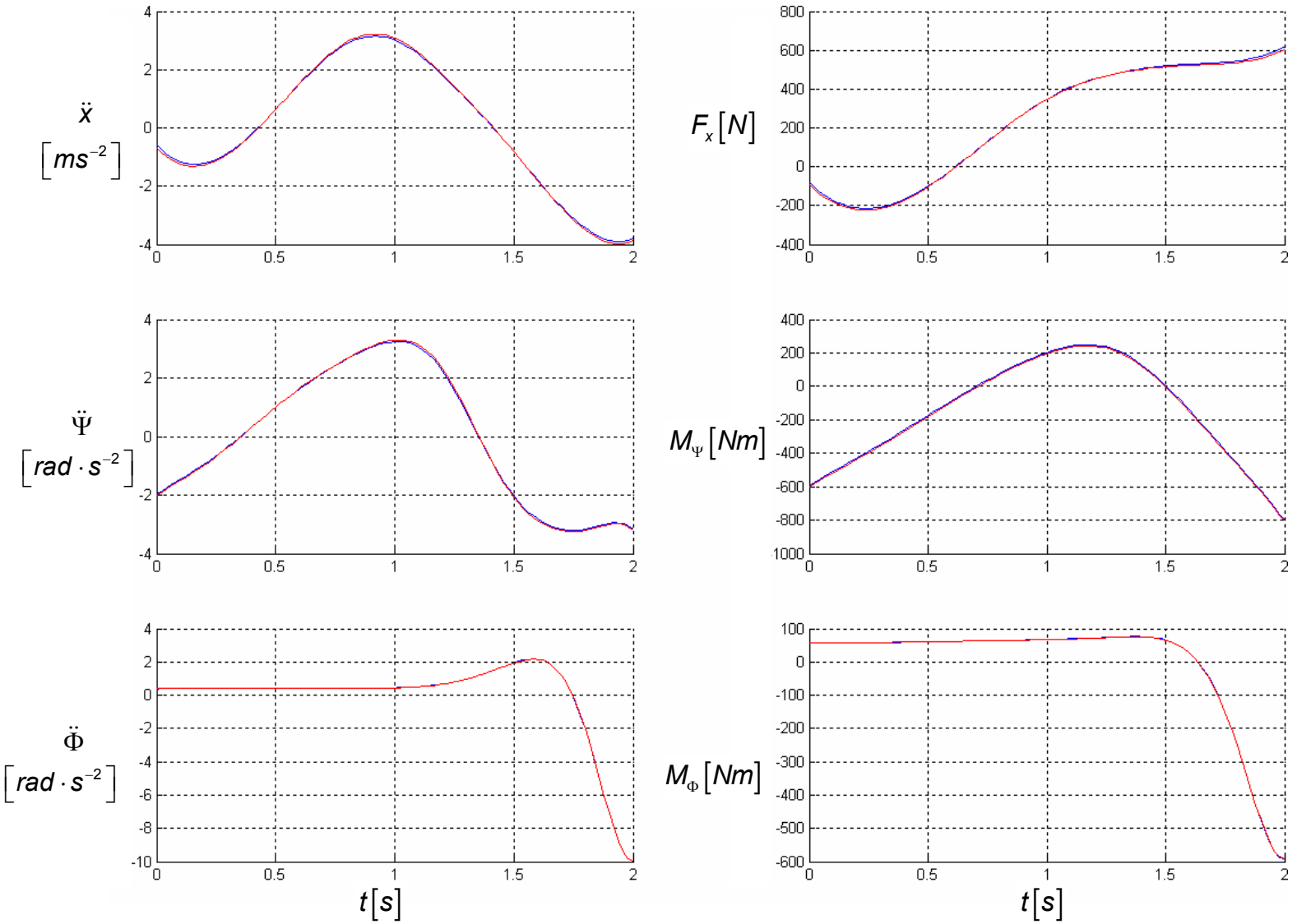
PŘÍLOHA 2C – porovnání řešení cylindrického robotu pomocí gradientní metody 1.řádu realizované v C++ a MATLAB



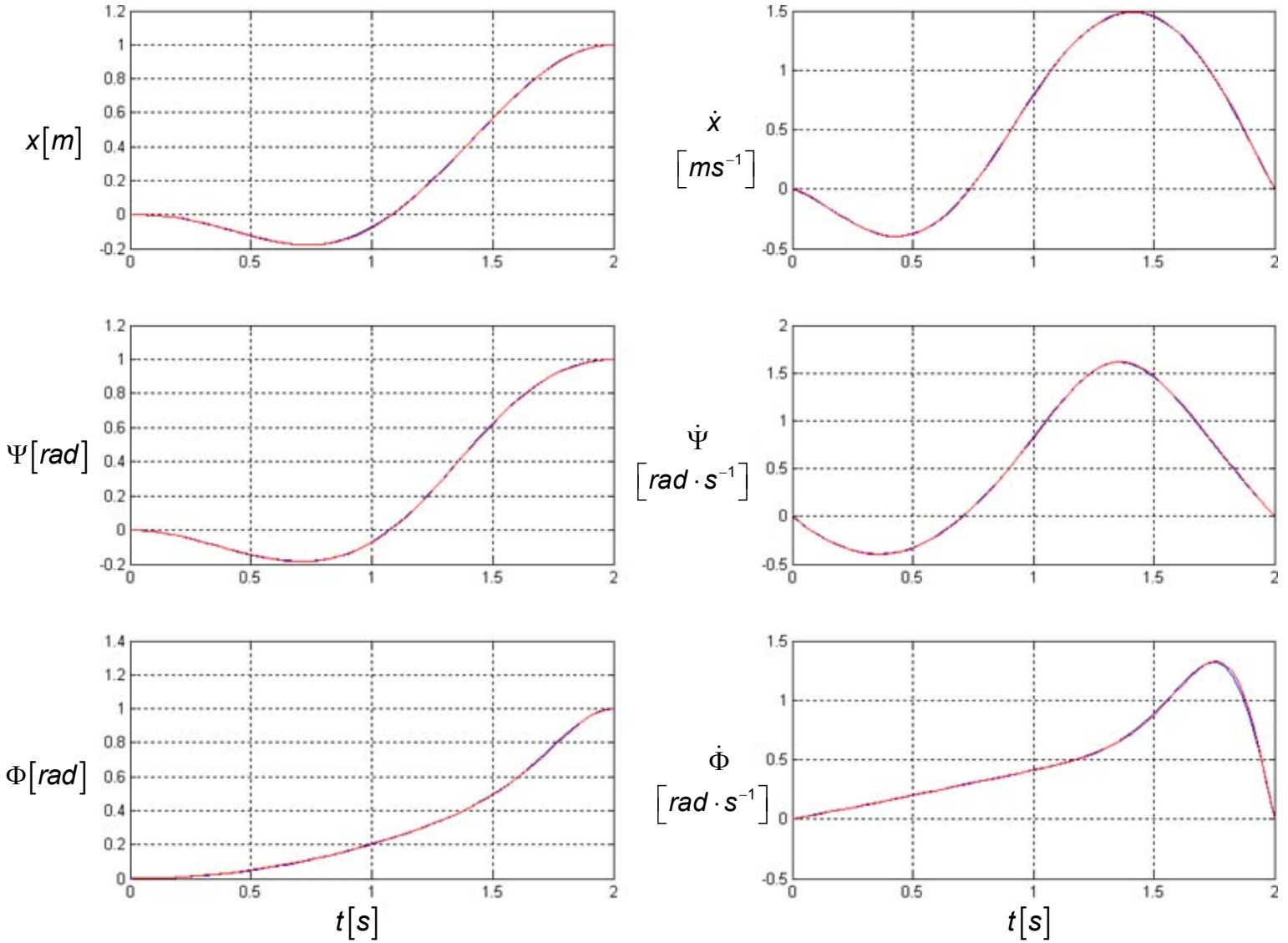
PŘÍLOHA 2D – porovnání řešení cylindrického robotu pomocí gradientní metody 1.řádu realizované v C++ a MATLAB



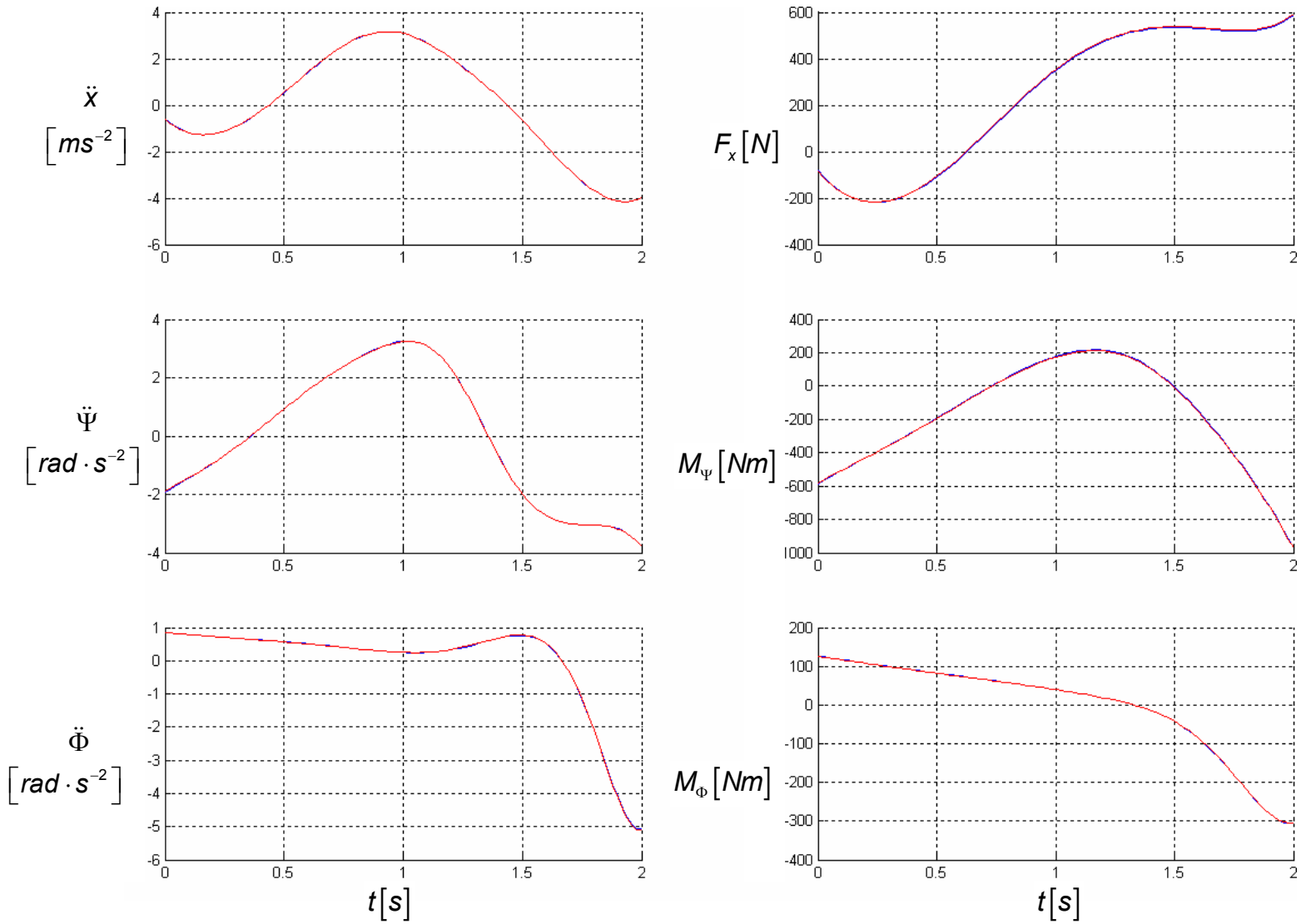
PŘÍLOHA 3A – porovnání řešení sférického robotu pomocí gradientní metody 1.řádu a metody nelineární střelby



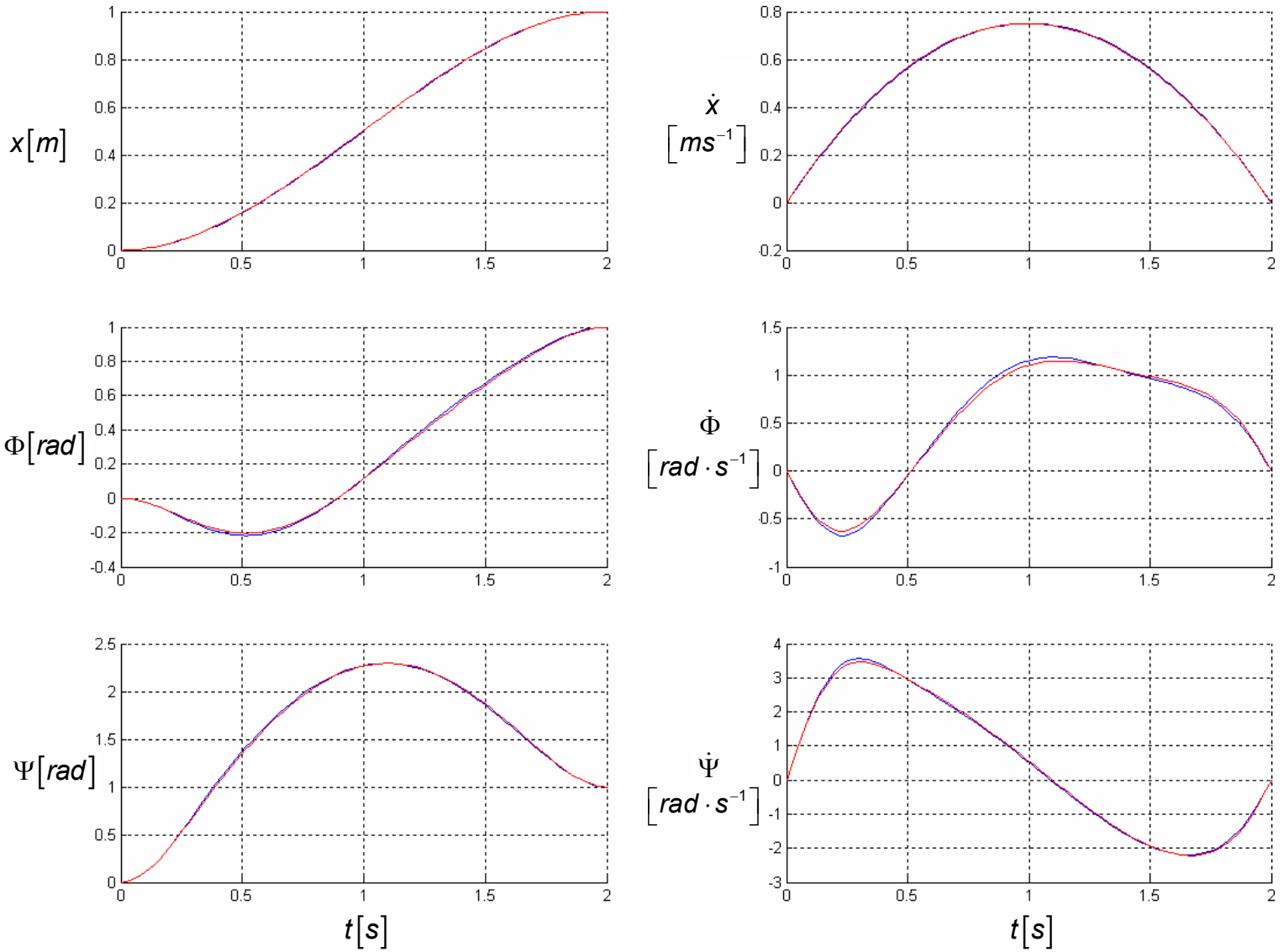
PŘÍLOHA 3B – porovnání řešení sférického robotu pomocí gradientní metody 1.řádu a metody nelineární střelby



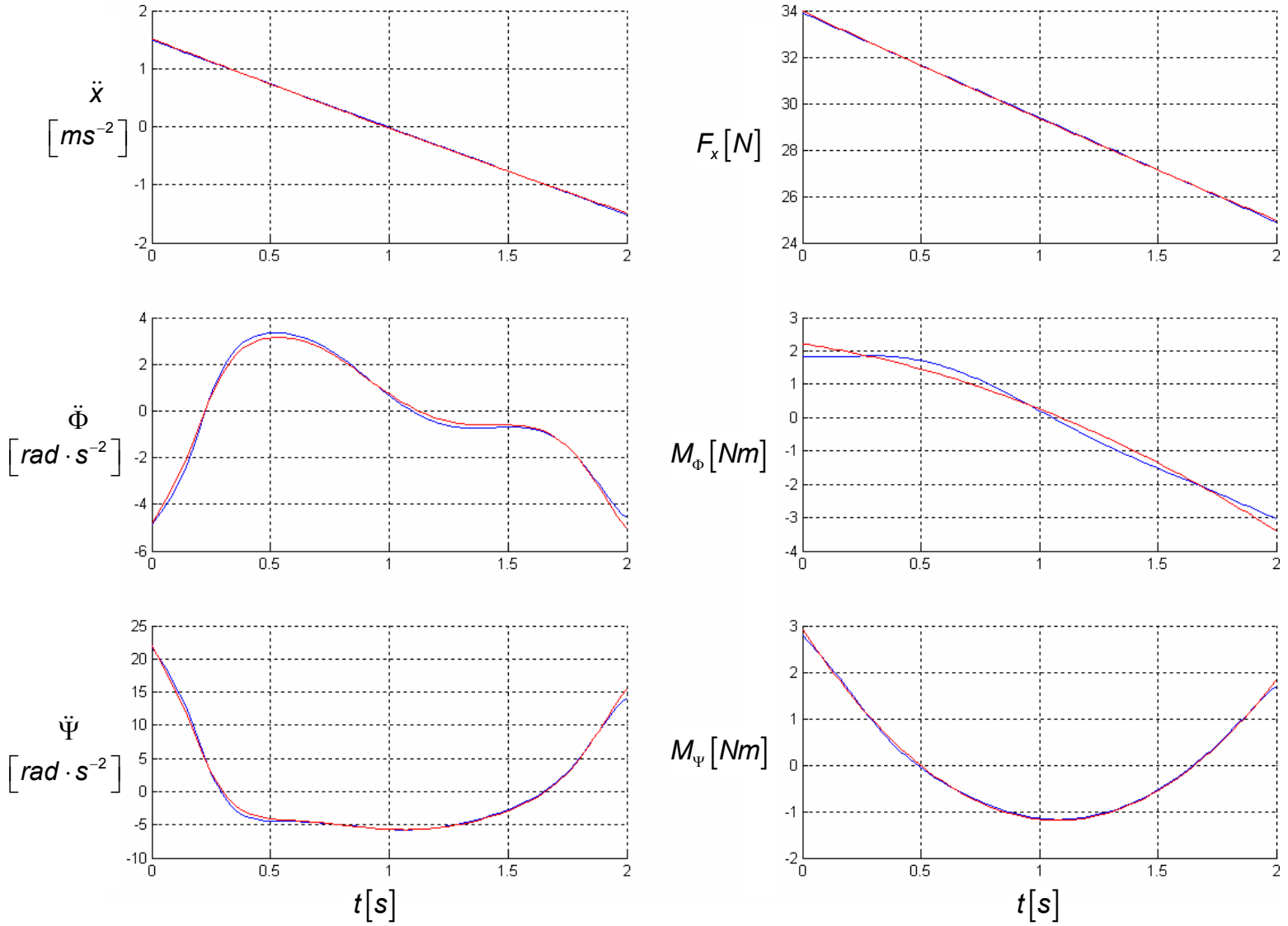
PŘÍLOHA 3C – porovnání řešení sférického robotu pomocí gradientní metody 1.řádu realizované v C++ a MATLAB



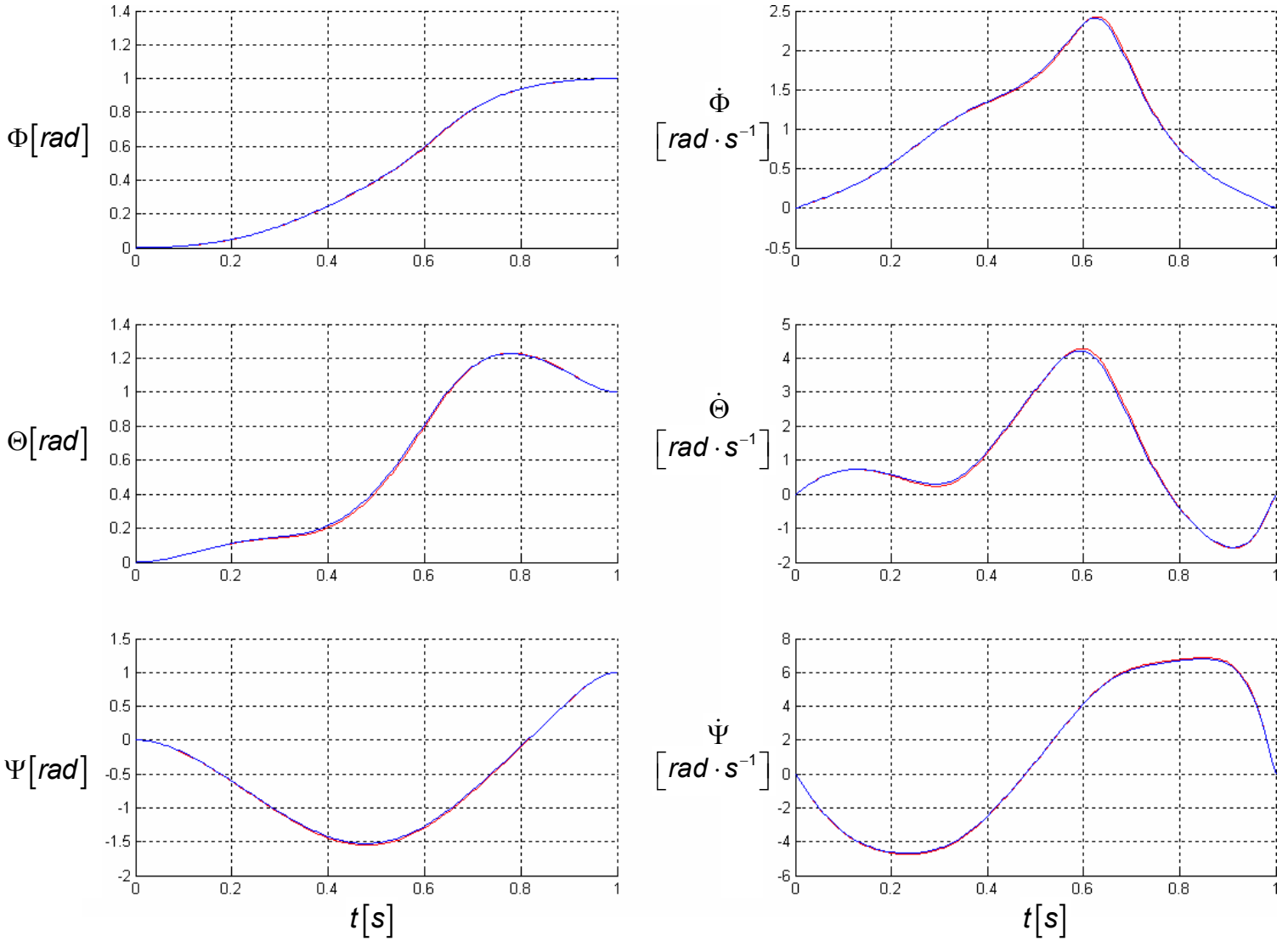
PŘÍLOHA 3D – porovnání řešení sférického robotu pomocí gradientní metody 1.řádu realizované v C++ a MATLAB



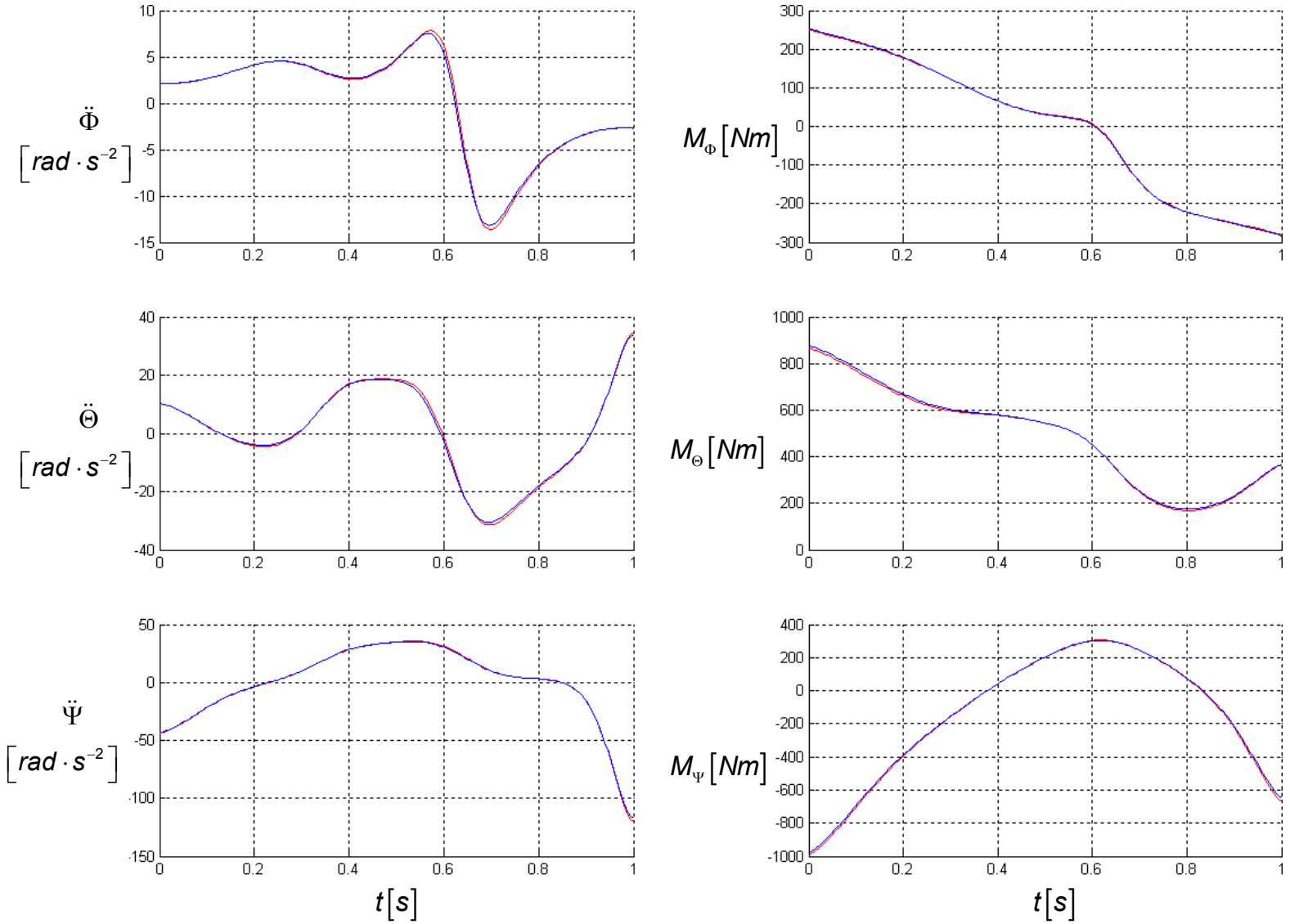
PŘÍLOHA 4A – porovnání řešení SCARA robotu pomocí gradientní metody 1.řádu a metody nelineární střelby



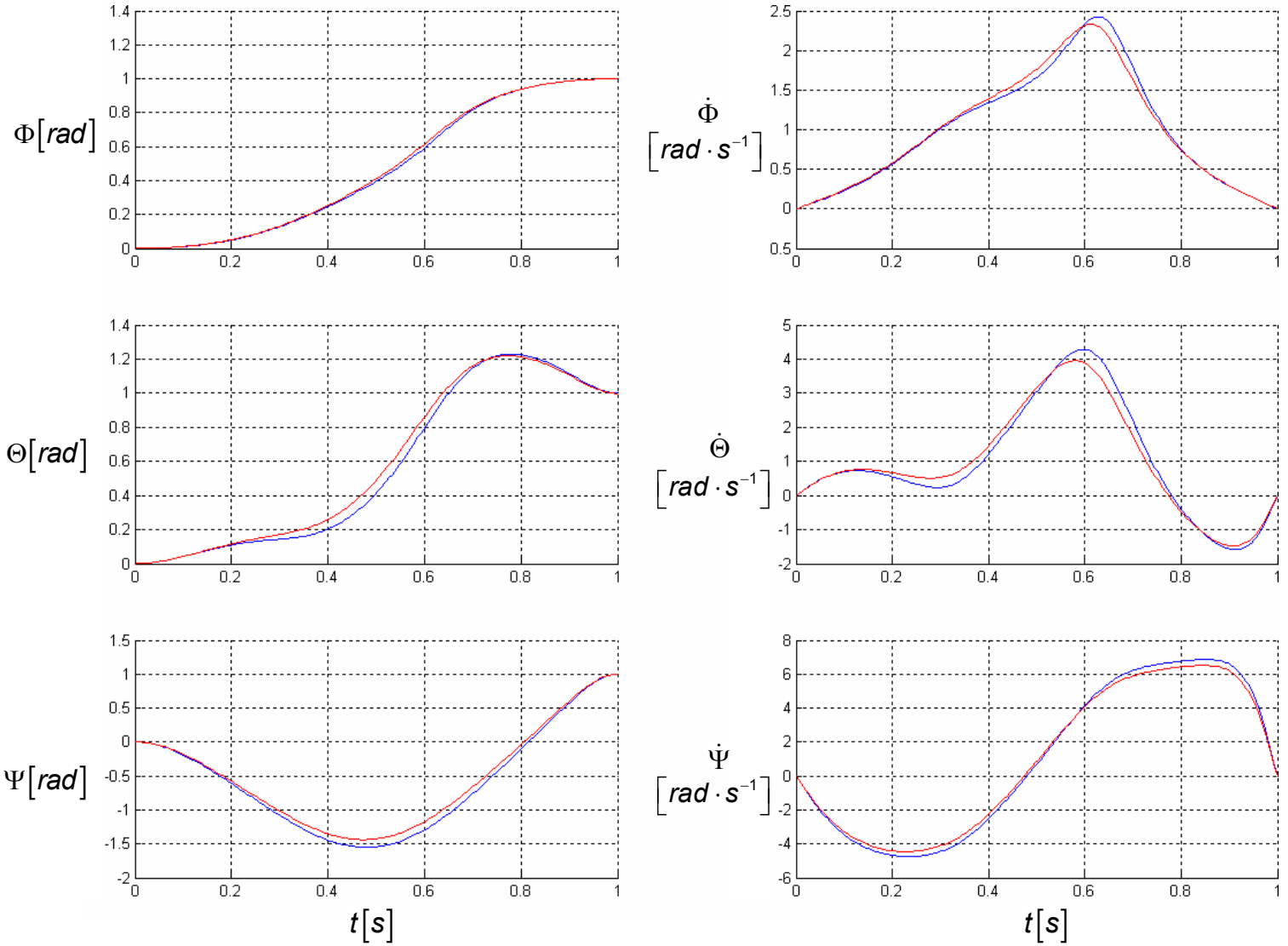
PŘÍLOHA 4B – porovnání řešení SCARA robotu pomocí gradientní metody 1.řádu a metody nelineární střelby



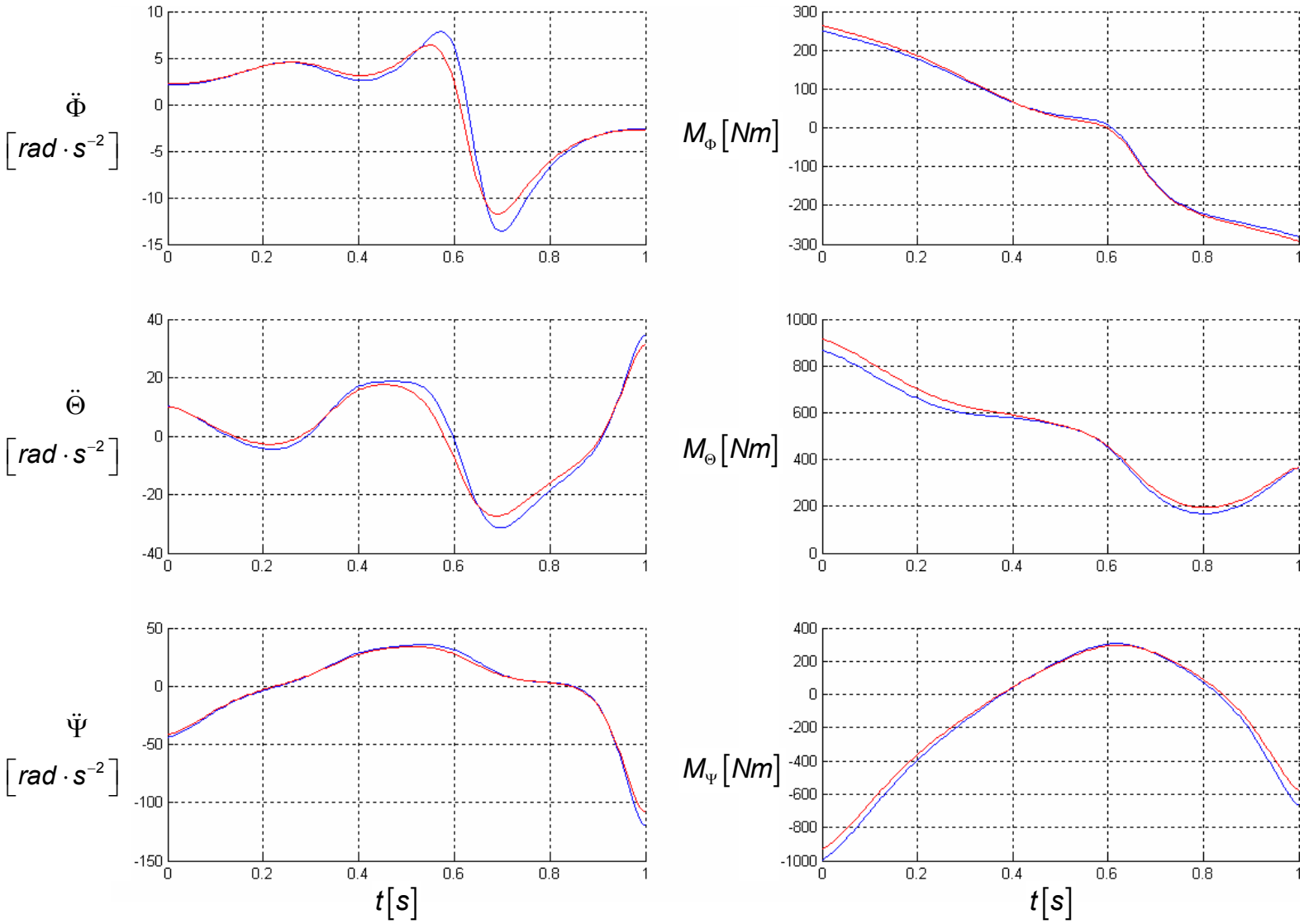
PŘÍLOHA 5A – porovnání řešení angulárního robotu pomocí gradientní metody 1.řádu a metody nelineární střelby



PŘÍLOHA 5B – porovnání řešení angulárního robotu pomocí gradientní metody 1.řádu a metody nelineární střelby



PŘÍLOHA 5C – porovnání řešení angulárního robotu pomocí gradientní metody 1.řádu realizované v C++ a MATLAB



PŘÍLOHA 5D – porovnání řešení angulárního robotu pomocí gradientní metody 1.řádu realizované v C++ a MATLAB

Technická univerzita v Liberci

Fakulta mechatroniky a mezioborových inženýrských studií

Katedra softwarového inženýrství

Příloha 6

Manuál k programu „Vybrané algoritmy optimálního řízení robotů“

User manual
„Selected algorithms of optimum robot control“

Liberec 2004 Jakub Štilec

Obsah

1. Instalace aplikace

Doplňkovou aplikaci k diplomové práci „Vybrané algoritmy optimálního řízení robotů“ můžete nainstalovat z přiloženého CD-ROM. Po vložení CD do mechaniky se automaticky spustí instalační program. V případě, že by nedošlo k jeho automatickému spuštění, spusťte program *setup**setup.exe*.

V instalačním programu lze nastavit adresář do kterého chcete aplikaci nainstalovat. Tento program také vytvoří zástupce aplikace a to jednak na ploše a také v nabídce Start, ve složce programy. Aplikaci je možné odstranit opětovným spuštěním instalačního programu, nebo z ovládacích panelů v nabídce přidat, nebo odebrat programy.

2. Základní struktura a ovládání aplikace

Aplikace je se skládá z jednotlivých modulů. Kliknutím na odkaz aplikace (vytvořeném při instalaci) se spustí soubor *CentralControl.exe*. Tato aplikace zprostředkovává spouštění jednotlivých modulů prostředí. Tyto moduly jsou obsaženy v podadresáři hlavního adresáře aplikace s názvem *bin*. Dále je pak možné z této aplikace měnit parametry jednotlivých řešených úloh. Tyto parametry jsou uloženy v podadresáři *CtrlData* v souborech s příponou *dat*. V tomto podadresáři jsou také uloženy virtuální modely jednotlivých částí robotů a to v souborech s příponou *d3d*. Poslední podadresář má název *SimData* a obsahuje řešení získaná z jednotlivých modulů, která jsou uložená v souborech s příponou *dat*. Součástí instalace aplikace je i několik ukázek získaných optimálních řešení základních struktur robotů.

Na obrázku 2.1 můžete nalézt okno aplikace. Výběr je tvořen pomocí karet se záložkami. Po výběru první záložky s označením *Gradientní metoda* se v její kartě zobrazí pět tlačítek. Stisknutím jednoho z tlačítek je spuštěn modul pro řešení dané úlohy pomocí gradientní metody 1. řádu. Těchto modulů je pět a umožňují řešení cylindrického, sférického, angulárního a SCARA robotu, nebo také jednoduché úlohy určené pro ověření funkce této metody.

Druhá záložka má označení *Nelineární střelba*. Po jejím výběru se opět zobrazí pět tlačítek. Stisknutím jednoho z nich je spuštěn modul pro řešení dané úlohy pomocí metody nelineární střelby. Tyto moduly umožňují řešení stejných typů úloh jako gradientní metoda 1. řádu.

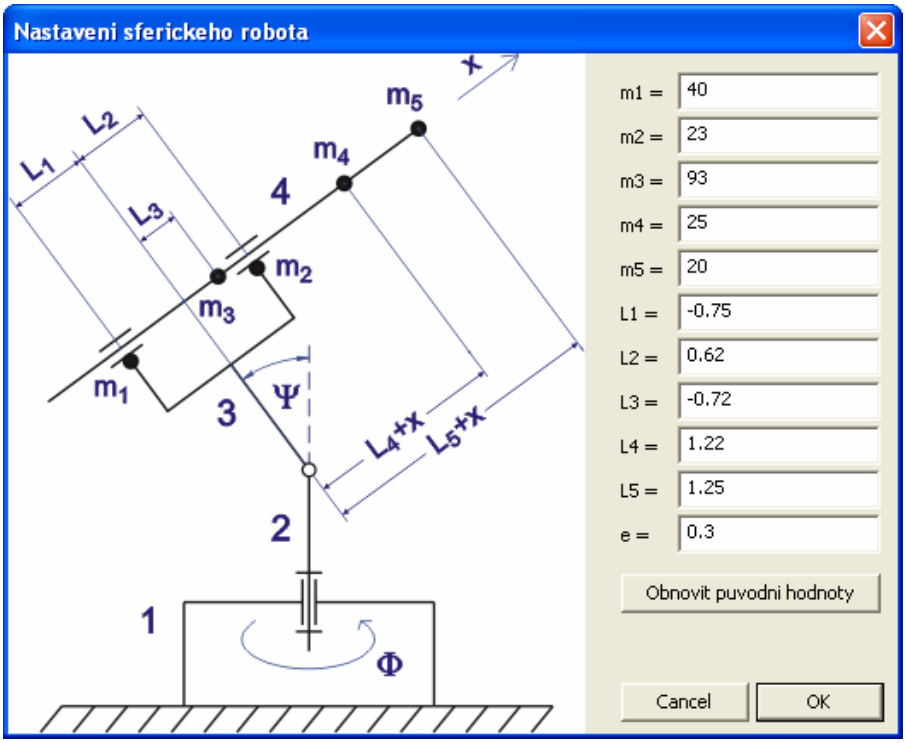
Třetí záložka je označena jako *Řízení s překážkou*. Po jejím označení se zobrazí pouze jedno tlačítko. Jeho stisknutím je spuštěn modul pro optimální řízení jednoduché úlohy v prostoru se statickou překážkou tvaru kružnice.

Obr. 2.1: Okno řídící části aplikace.

Poslední záložka je označena jako *Nastavení*. Po jejím výběru je zobrazeno pět tlačítek. Stisknutím libovolného z nich je zobrazeno dialogové okno, které umožňuje nastavit parametry cylindrického, sférického, angulárního a SCARA robotu a také jednoduché úlohy. Ukázku dialogového okna pro nastavení parametrů sférického robotu můžete nalézt na obrázku 2.2. V levé části okna je zobrazeno schéma daného robotu, v pravé části pak seznam jednotlivých parametrů. Pro obnovení výchozích parametrů slouží tlačítko *Obnovit původní hodnoty*.

Zaškrtávací políčko *Poloha okna* nad všemi ostatními, umožňuje umístění hlavního okna aplikace nad všechny ostatní okna. To je výhodné zejména tehdy pokud spouštíme více modulů najednou, protože hlavní okno aplikace je stále dostupné.

Tlačítko označené jako *VR prohlížeč* umožňuje spuštění modulu pro zobrazení vypočítaných optimálních průběhů pohybů základních struktur robotů ve 3D a také formou 2D řezů.

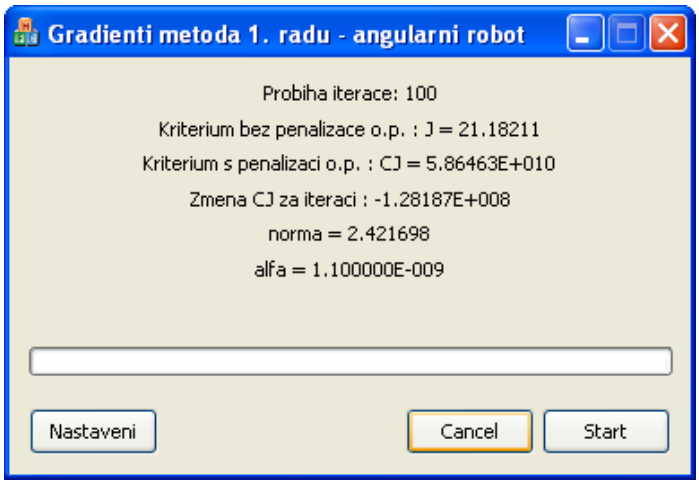


Obr. 2.2: Okno nastavení parametrů sférického robotu.

3. Popis modulu gradientní metody 1. řádu

Jednotlivé moduly pro řešení úloh optimálního řízení, pomocí gradientní metody 1. řádu, se příliš neliší, a proto si jejich ovládání popíšeme na konkrétním modulu pro řešení robotu pracujícího v angulárním souřadném systému. Okno tohoto modulu můžete vidět na obrázku 3.1.

Výpočet je spuštěn pomocí tlačítka *Start*. Je samozřejmě možné výpočet kdykoliv přerušit pomocí tlačítka *Cancel*. Po stisku tohoto tlačítka se získané výsledky zobrazí jako průběhy poloh, rychlostí, zrychlení a řízení jednotlivých členů a dále se uloží do souboru definovaného v nastavení. Průběhy řízení zůstávají uchovány v paměti a po opětovném spuštění výpočtu jsou brány jako počáteční odhad. Tak je možné výpočet kdykoliv přerušit, změnit parametry a ve výpočtu dále pokračovat beze ztráty dat.

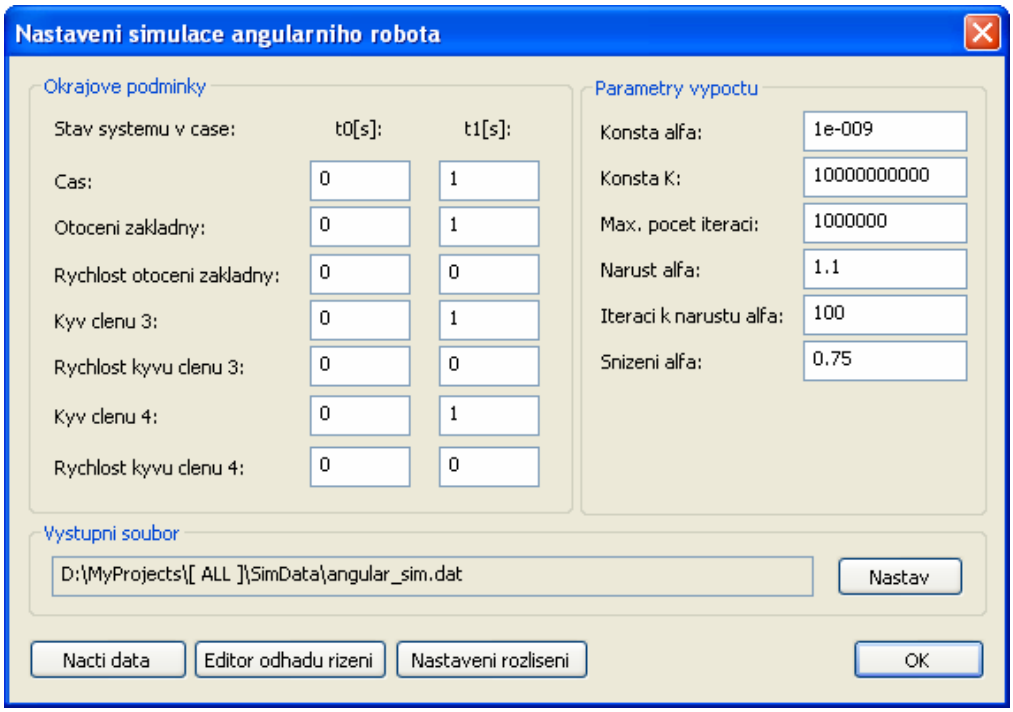


Obr. 3.1: Okno modulu gradientní metody 1. řádu.

Popišme parametry zobrazované v dialogovém okně uvedeném na obr. 3.1. Horní textové pole zobrazuje číslo právě probíhající iterace, ve druhém řádku je obsažena hodnota funkce kritéria *J* bez penalizace splnění okrajových podmínek. V následujícím řádku je pak hodnota funkce kritéria s penalizací splnění okrajových podmínek *CJ* a čtvrtý řádek obsahuje změnu této hodnoty během právě proběhlé iterace. Předposlední řádek zobrazuje normu odchylky od splnění okrajových podmínek a poslední řádek obsahuje aktuální hodnotu konstanty α.

Selhání výpočtu je reprezentováno zobrazením hodnoty *1.#QNAN0* v některém ze zobrazovacích polí. V takovémto případě je třeba výpočet přerušit, nastavit nový lepší počáteční odhad, resp. jiné parametry výpočtu a výpočet znovu spustit.

Stisknutím tlačítka *Nastavení* se zobrazí dialogové okno nastavení parametrů výpočtu. Toto okno je na obrázku 3.2. Zde je možné nastavit počáteční a koncové okrajové podmínky požadovaného řešení úlohy. Stisknutím tlačítka *Nastav* je uživateli umožněno nastavit výstupní soubor, do kterého jsou ukládány výsledky řešení. Pokud již tento soubor existuje, je automaticky přepisován, vždy při ukončení výpočtu. V případě, že soubor není možné zapsat, je uživatel upozorněn varovnou hláškou.



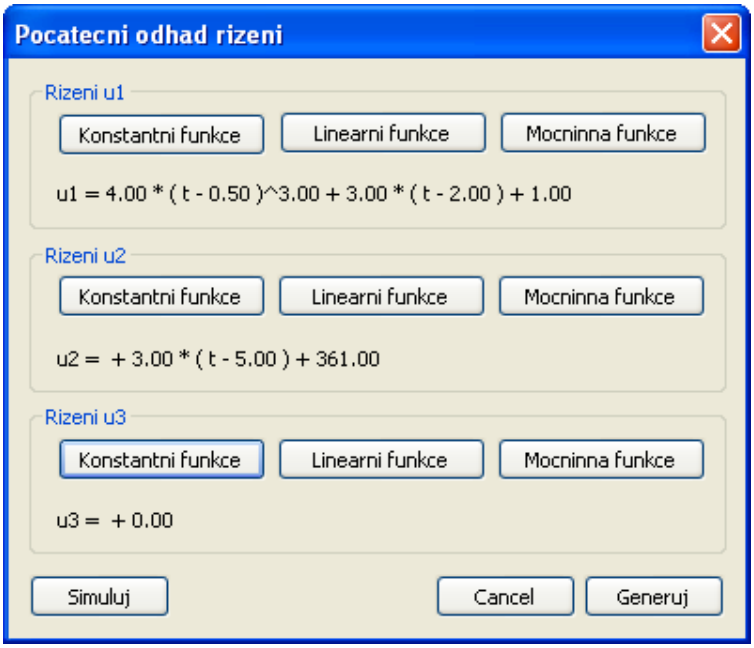
Obr. 3.2: Okno nastavení parametrů výpočtu gradientní metody 1. řádu.

V parametrech výpočtu je možné nastavit počáteční odhad konstanty α a velikost penalizace splnění okrajových podmínek *K*. Dále koeficient, kterým se α násobí po uplynutí *n* iterací a koeficient, kterým se násobí při chybě výpočtu, tedy pokud v dané iteraci není dosaženo záporné změny *CJ*. V takovém případě se obnoví původní hodnota řízení, kdy *CJ* bylo záporné, α se sníží a výpočet pokračuje.

Tlačítko *Načti data* umožňuje načíst okrajové podmínky, rozlišení a průběh řízení v čase ze souboru typu dat, který je programem ukládán v průběhu výpočtu. Lze načítat pouze data shodných typů úloh, v opačném případě je uživatel upozorněn varovnou hláškou.

Tento typ souboru je kompatibilní se soubory generovanými moduly pro řešení úloh pomocí metody nelineární střelby a tak je možné hodnoty získané pomocí této metody použít jako počáteční odhad pro gradientní metodu 1. řádu.

Tlačítko *Editor odhadu řízení* spustí další dialogové okno, které uživateli umožňuje generovat počáteční odhad řízení. Pro generování jeho jednotlivých složek jsou k dispozici tři základní funkce, ze kterých je možné výsledné řízení skládat. Jedná se o konstantní funkci, lineární funkci a mocninnou funkci. Pro účely této práce se použité funkce jeví jako plně dostačující. Hodnoty daných funkcí jsou zadány kliknutím na příslušné tlačítko a jejich nastavením v zobrazeném dialogovém boxu. Je zde také k dispozici tlačítko *Simuluj*, které umožňuje ověřit, jak se systém pro zvolené řízení chová. Okno editoru odhadu řízení je zobrazeno na následujícím obrázku.



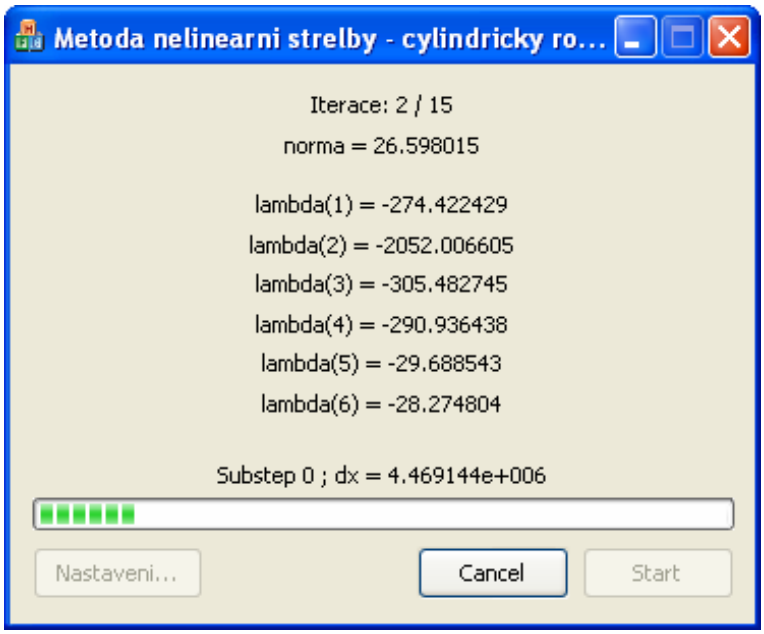
Obr. 3.3: Editor odhadu řízení.

V nastavení parametrů výpočtu gradientní metody 1. řádu ještě zbývá popsat funkci posledního tlačítka s označením *Nastavení rozlišení*. Po jeho stisknutí je zobrazeno dialogové okno, ve kterém je možné nastavit počet bodů rozlišení simulace. Po změně této hodnoty je uživatel dotázán zda se má řízení uložené v paměti interpolovat na nové

rozlišení (odpověď NE), nebo zda se má vygenerovat nové řízení z parametrů uložených v editoru odhadu řízení (odpověď ANO) .

4. Popis modulu metody nelineární střelby

Ovládání modulů pro řešení úloh optimálního řízení pomocí metody nelineární střelby si opět popíšeme na konkrétním modulu, určeném pro řešení robotu pracujícího v cylindrickém souřadném systému. Okno tohoto modulu můžete vidět na následujícím obrázku.



Obr. 4.1: Okno modulu metody nelineární střelby.

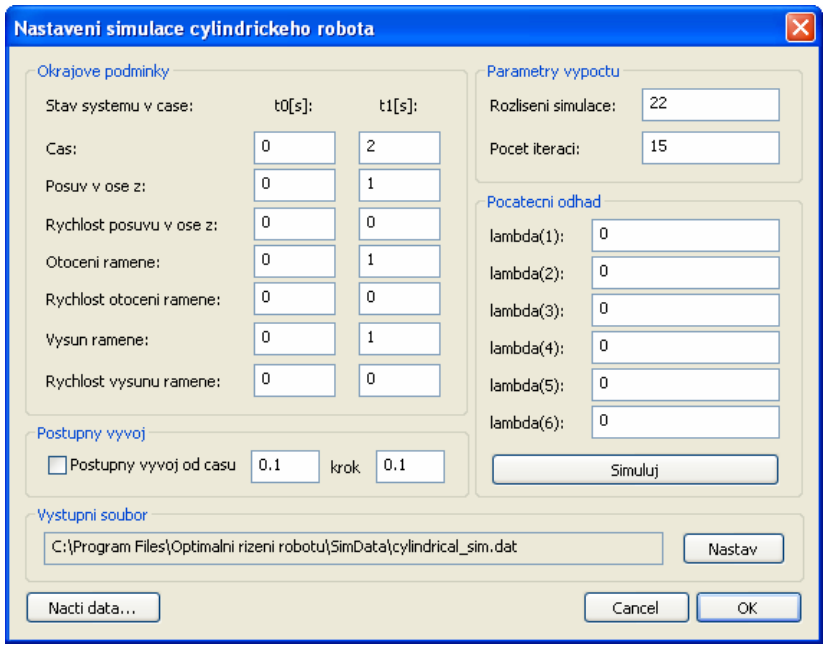
Výpočet je spuštěn pomocí tlačítka *Start*. Je samozřejmě možné výpočet kdykoliv přerušit pomocí tlačítka *Cancel*. Po stisku tohoto tlačítka se získané výsledky zobrazí jako průběhy poloh, rychlostí, zrychlení a řízení jednotlivých členů a dále se také uloží do souboru definovaného v nastavení. Hodnoty vektoru λ zůstávají uchovány v paměti a po opětovném spuštění výpočtu jsou brány jako počáteční odhad. Tak je možné výpočet kdykoliv přerušit, změnit parametry a ve výpočtu dále pokračovat beze ztráty dat.

Popišme parametry zobrazované v tomto dialogovém okně. Horní textové pole zobrazuje číslo právě probíhající iterace a celkový počet iterací. Ve druhém řádku je obsažena norma odchylky od splnění okrajových podmínek. Následující řádky obsahují informace o aktuálních hodnotách jednotlivých složek vektoru λ. Poslední řádek obsahuje číslo sub kroku. Jedná se o přidání dalších kroků v rámci dané iterace tak, aby byla dosáhnuta požadovaná přesnost řešení. Hodnota *dx* udává velikost změny vektoru λ. Pokud se v tomto poli objeví hláška *Error*, pak nebylo možné během dané iterace najít takovou změnu vektoru λ tak, aby byla splněna požadovaná přesnost Výpočet pak s nejvyšší pravděpodobností během několika iterací selže.

Selhání výpočtu je reprezentováno zobrazením hodnoty *1.#QNAN0* v některém ze zobrazovacích polí. V takovém případě je třeba výpočet přerušit, nastavit nové počáteční hodnoty vektoru λ a výpočet znovu spustit.

Stisknutím tlačítka *Nastavení* se zobrazí dialogové okno nastavení parametrů výpočtu. Toto okno je zobrazeno na obrázku 4.2. Zde je možné nastavit počáteční a koncové okrajové podmínky požadovaného řešení úlohy. Stisknutím tlačítka *Nastav* je uživateli umožněno nastavit výstupní soubor, do kterého jsou ukládány výsledky řešení. Pokud již tento soubor existuje, je automaticky přepisován, vždy při ukončení výpočtu. V případě, že soubor není možné zapsat, je uživatel upozorněn varovnou hláškou.

V parametrech výpočtu je možné nastavit počet bodů rozlišení simulace a také celkový počet iterací. Dále je zde možné nastavit funkci postupného časového vývoje řešení. Zde se nastaví počáteční čas a krok růstu tohoto času. Výpočet je pak prováděn v cyklech, kdy koncový čas řešení je nastaven na počáteční čas časového vývoje plus krok růstu času. V jednotlivých iteracích se vždy nalezne řešení pro daný koncový čas. Tento koncový čas je pak zvýšen o hodnotu kroku růstu času a takto výpočet pokračuje až do dosažení řešení pro zadaný koncový čas v okrajových podmínkách.



Obr. 4.2: Okno nastavení parametrů výpočtu metody nelineární střelby.

Tlačítko *Načti data* umožňuje načíst okrajové podmínky a rozlišení ze souboru typu dat, který je programem ukládán v průběhu výpočtu. V případě, že byl tento soubor generován z modulu pro řešení úlohy pomocí metody nelineární střelby, pak je z něj načtena také hodnota vektoru λ. Lze načítat pouze data shodných typů úloh, v opačném případě je uživatel upozorněn varovnou hláškou.

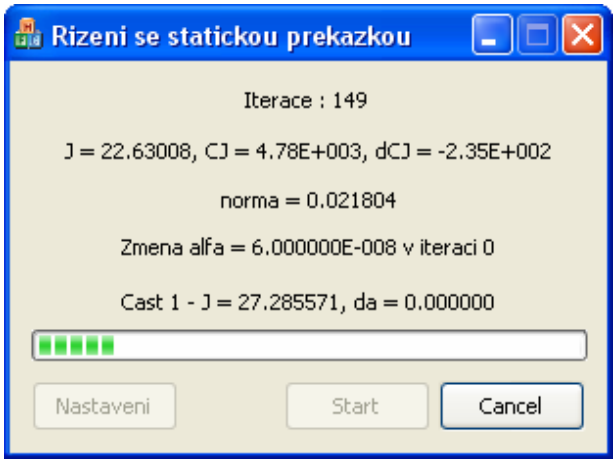
V dialogovém okně lze také nastavit počáteční odhad vektoru λ a to číselným nastavením jeho jednotlivých složek v textových polích. Tlačítkem *Simuluj* je pak možné ověřit chování a stabilitu systému pro zvolený vektor λ.

5. Popis modulu pro řešení úlohy se statickou překážkou

Dialogové okno tohoto modulu můžete vidět na obrázku 5.1. Výpočet lze spustit pomocí tlačítka *Start,* přerušení výpočtu je možné pomocí tlačítka *Cancel*. Po stisku tohoto tlačítka před dokončením výpočtu se zobrazí neúplné výsledky. Výstupem modulu je optimální průběh trajektorie hmotného bodu a to jednak bez ohledu na statickou překážku

(znázorněno zelenou barvou) a dále s ohledem tuto překážku (znázorněno červenou barvou).

Popišme parametry zobrazované v tomto dialogovém okně. Horní textové pole zobrazuje číslo probíhající iterace. V následujícím řádku jsou obsaženy hodnoty funkce kritéria *J* bez penalizace splnění okrajových podmínek, funkce kritéria s penalizací splnění okrajových podmínek *CJ* a změna hodnoty *CJ* během právě proběhlé iterace označená jako *dCJ*. Ve třetím řádku je zobrazena norma odchylky od splnění okrajových podmínek. Následující řádek obsahuje informaci o úpravě hodnoty konstanty α v *i*-té iteraci. A poslední textové pole zobrazuje průběh výpočtu. Tedy, zda je prováděn počáteční odhad, nebo výpočet první, resp. třetí části trajektorie. Dále je zde uvedena hodnota kritéria *J* v koncovém, resp. počátečním bodě posunutém od průsečíku s kružnicí o úhel *da*. Takto je vhodně znázorněn průběh vyhledávaní optimálního napojení na kružnici.

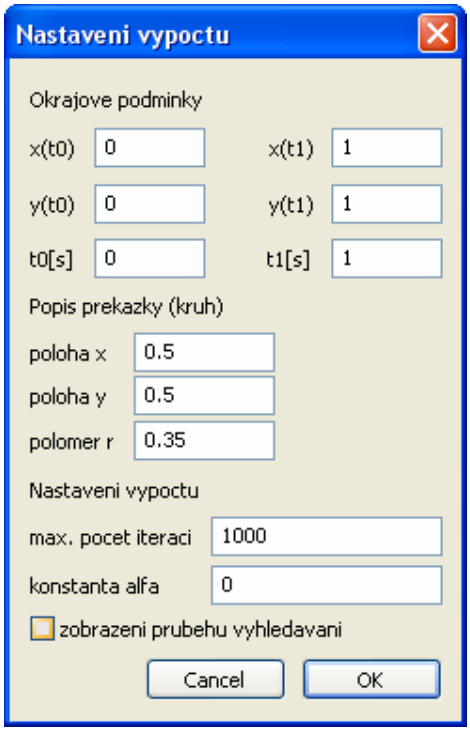


Obr. 5.1: Okno modulu pro řešení úlohy se statickou překážkou.

Selhání výpočtu je reprezentováno zobrazením hodnoty *1.#QNAN0* v některém ze zobrazovacích polí. V takovém případě je třeba výpočet přerušit, nastavit vhodnější parametry výpočtu a výpočet znovu spustit.

Stisknutím tlačítka *Nastavení* se zobrazí dialogové okno nastavení parametrů výpočtu. Toto okno je zobrazeno na obrázku 5.2. Zde je možné nastavit počáteční a koncovou polohu hmotného bodu a také počáteční a koncový čas řešení úlohy. Uživateli zde také může nastavit parametry překážky typu kruh, tedy polohu středu a poloměr. Dále je zde

nastavení maximálního počtu iterací, během jednoho kroku vyhledávání a velikost konstanty α.



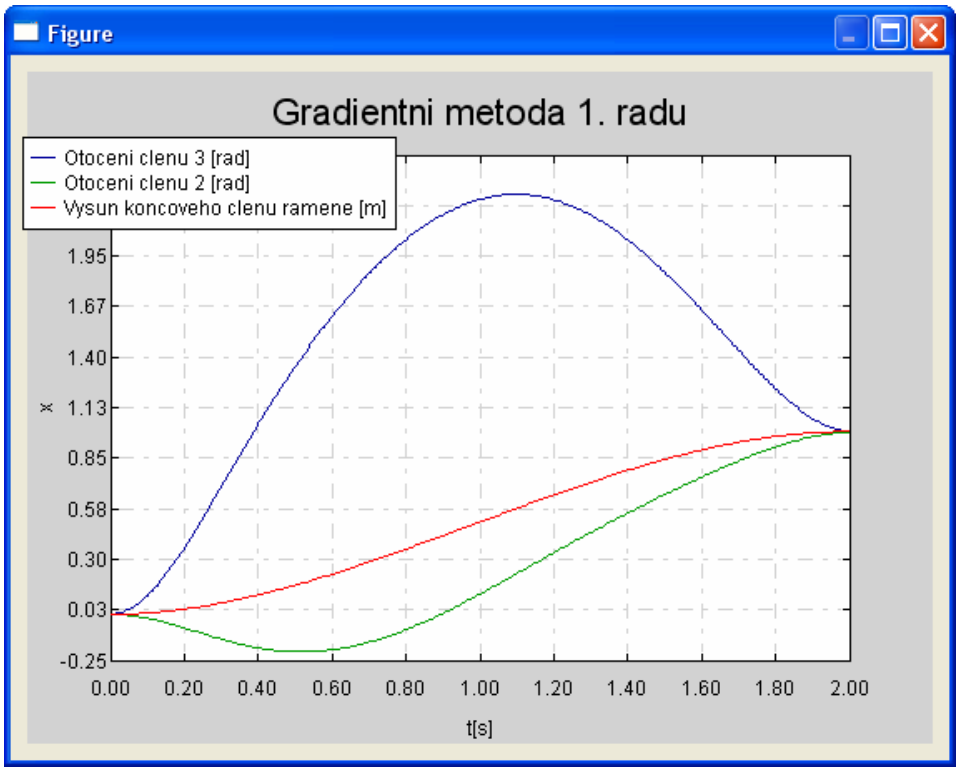
Obr. 5.2: Okno nastavení parametrů výpočtu metody pro řešení úlohy se statickou překážkou.

6. Popis možností zobrazení výsledných průběhů

Výstupem jednotlivých modulů této aplikace jsou průběhy polohy, rychlostí, zrychlení a řízení všech členů řešených systémů. Proto v této části manuálu popíšeme možnosti jejich zobrazení.

Okno zobrazení je na obrázku 6.1. Legendu je možné libovolně přesouvat. Červenou barvou je označena křivka číslo 1, zelenou barvou křivka číslo 2 a modrou barvou křivka číslo 3. Volba možnosti zobrazení je vytvořena přes kontextové menu, přístupné pomocí kliknutí pravým tlačítkem myši uvnitř oblasti okna. Toto menu je znázorněno na obrázku 6.2. Je zde možné zobrazit průběhy poloh, rychlostí, zrychlení, nebo řízení všech členů

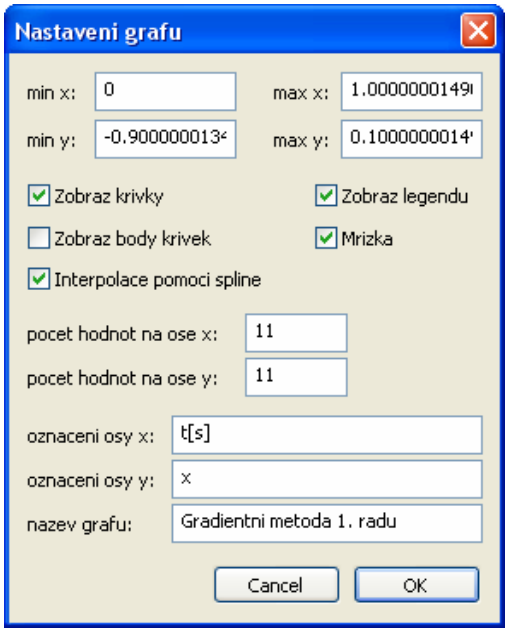
robotu. V případě potřeby je možné zobrazit pouze zvolený průběh jednoho z členů a to kliknutím na volbu v menu *Křivka 1,2, nebo 3*.



Obr. 6.1: Okno pro zobrazení vypočtených průběhů.

Obr. 6.2: Kontextové menu volby zobrazení.

Pro další nastavení zobrazení je možné zvolit volbu *Nastavení* v menu. Výběr této volby otevře dialogové okno, které lze nalézt na obrázku 6.3. V tomto okně je možné nastavit minimální a maximální hodnoty os *x* a *y*. Dále zde uživatel může nastavit zobrazení legendy, jednotlivých křivek, uzlových bodů těchto křivek a mřížky. Dále zde lze vypnout interpolace uzlových bodů pomocí kubické spline křivky (uzlové body pak budou spojeny pomocí úseček). Uživatel také může nastavit počet číselných hodnot, které se mají zobrazovat na osách *x* a *y*, popisky těchto os a název celého zobrazení.



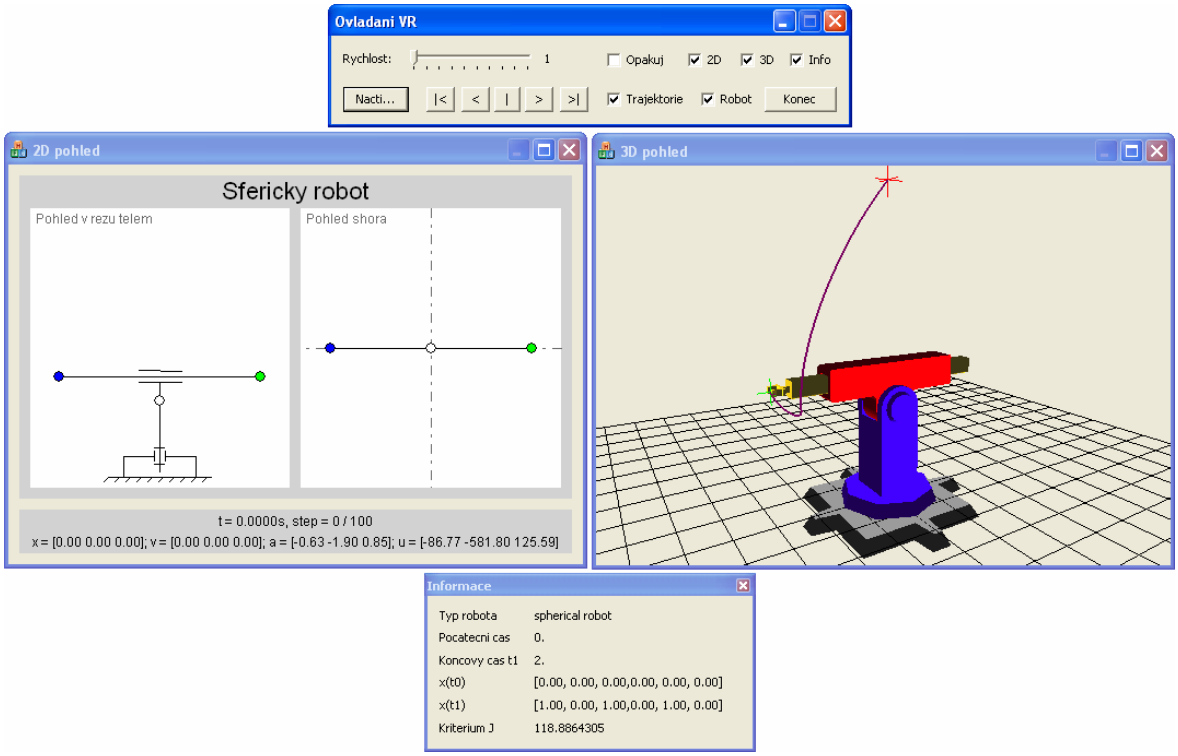
Obr. 6.3: Nastavení zobrazení.

7. Popis modulu virtuálního zobrazení

Na následujícím obrázku 7.1 můžete nalézt prostředí tohoto modulu. Skládá se ze čtyř oken. Tato okna jsou vzájemně svázána a pohybem horního okna se společně pohybují i okna ostatní. V následujícím textu jsou psány funkce jednotlivých oken.

První z nich s názvem *Ovládání VR* slouží pro řízení přehrávání scény a nastavení zobrazení. Pomocí tlačítka *Načti* je možné načíst soubor obsahující data získaná z modulů pro řešení pomocí gradientní metody 1. řádu, nebo metody nelineární střelby. Tento modul umožňuje zobrazení cylindrického, sférického, angulárního a SCARA robotu.

Druh robotu je podle dat obsažených v souboru automaticky rozpoznán a podle něho je pak nastavena celá scéna. Pomocí posuvníku s označením *rychlost* je možné nastavit rychlost přehrávání v rozsahu jedna až deset. Dále je zde pět tlačítek pro řízení přehrávání. Jednotlivá tlačítka mají po řadě následující význam: přejdi na počátek, přehrávej ve zpětném směru, zastav přehrávání, přehraj, přejdi na konec. Pomocí zaškrtávacích polí je možné nastavit opakování (přehrává se v normálním a zpětném směru v uzavřené smyčce), zobrazení 2D pohledu, zobrazení 3D pohledu, zobrazení informačního okna, dále pak zobrazení trajektorie efektoru a těla robotu v 3D pohledu.



Obr. 7.1: Okno modulu virtuálního zobrazení.

Druhé okno slouží jako 2D zobrazení. Levá část obsahuje řez tělem robotu, u SCARA robotu se jedná o řez vedoucí rovinou prvního a druhého členu ramene. Pravá část okna obsahuje pohled na rameno robotu shora. Dolní část obsahuje informace o aktuálním času právě zobrazeného snímku a tomuto času odpovídajícím hodnotám poloh, rychlostí, zrychlení a řízení jednotlivých členů ramene robotu.

Třetí okno reprezentuje 3D zobrazení. Mřížkou je zde naznačen podklad ramene robotu. Fialová křivka zobrazuje trajektorii efektoru, kde zelený křížek vyjadřuje její počáteční bod, fialový křížek označuje aktuální polohu efektoru v čase a červený křížek reprezentuje koncový bod trajektorie. Pohyb myši spolu se stisknutým levým tlačítkem způsobuje rotaci scény. Pohyb myši spolu se stisknutým pravým tlačítkem způsobuje posunutí scény. Pomocí kláves +, nebo – na numerické klávesnici je možné provádět změnu měřítka zobrazované scény.

Poslední okno obsahuje informace o načteném řešení. Je zde uveden název struktury robotu, počáteční a koncový čas řešení. Dále hodnota stavového vektoru v počátečním a koncovém čase a hodnota kritéria *J* získaného řešení.